AFRL-RI-RS-TR-2010-080
Final Technical Report
March 2010

SIMULATION FOR DYNAMIC SITUATION AWARENESS AND PREDICTION III

Northrop Grumman Space & Mission Systems Corp.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2010-080 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/
DAWN TREVISANI
Work Unit Manager

/s/
JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MARCH 2010	Final	March 2006 – September 2009

4. TITLE AND SUBTITLE

SIMULATION FOR DYNAMIC SITUATION AWARENESS AND PREDICTION III

5a. CONTRACT NUMBER
FA8750-06-C-0017

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Kevin Trott

5d. PROJECT NUMBER
459S

5e. TASK NUMBER
N6

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Northrop Grumman Space & Mission Systems Corp.
2340 Dulles Corner Blvd.
Herndon, VA 20171-3415

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2010-080

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2010-1107 Date Cleared: 12-March-2010

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This is a Final Technical Report documenting the Simulation for Dynamic Situation Awareness and Prediction (DSAP) III project, and the lessons learned during its performance. It is submitted by Northrop Grumman Corp. for the Air Force Research Laboratory (AFRL), Decision Support Systems Branch (RISB), under Contract FA 8750-06-C-0017, in accordance with CLIN 0002, CDRL A006. This report includes tasks such as the Joint Synthetic Battlespace for Research and Development (JSB-RD) distributed simulation environment and describes the scenarios that were generated and exercises using that capability; the JView Coordinate, Orientation, and Vector Conversion Services software; the Distributed Mission Operations (DMO) Test Harness (DMOTH) software; and the NSim simulation software, and how it was interfaced with the JSB-RD distributed simulation environment. Lessons learned, conclusions and recommendations are presented.

15. SUBJECT TERMS

Operationally focused simulation; Decision support; Distributed simulation environment, JView Coordinate, Orientation, and Vector Conversion Services software; Distributed Mission Operations (DMO) Test Harness (DMOTH).

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dawn Trevisani
U	U	U	UU	96	19b. TELEPHONE NUMBER (*Include area code*) N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This Final Technical Report documents the Simulation for Dynamic Situation Awareness and Prediction (DSAP) III project, and the lessons learned during its performance. It is submitted by Northrop Grumman for the Air Force Research Laboratory (AFRL), Decision Support Systems Branch (RISB), under Contract FA8750-06-C-0017, in accordance with CLIN 0002, CDRL A006.

The remainder of this report is organized as follows:

- Section 2 discusses the Joint Synthetic Battlespace for Research and Development (JSB-RD) distributed simulation environment, and the scenarios that were generated using that capability;
- Section 3 discusses the JView Coordinate, Orientation, and Vector Conversion Services software;
- Section 4 discusses the Distributed Mission Operations (DMO) Test Harness (DMOTH) software;
- Section 5 discusses the NSim simulation software, and how it was interfaced with the JSB-RD distributed simulation environment;
- Section 6 presents lessons learned, conclusions, and recommendations;
- Section 7 provides notes, including a glossary and an acronym list; and
- Section 8 lists the documents referenced in this report.

2 JOINT SYNTHETIC BATTLESPACE FOR RESEARCH AND DEVELOPMENT

This section discusses the Joint Synthetic Battlespace for Research and Development (JSB-RD), a simulation capability assembled from a number of existing tools and components. It is organized as follows:

- Section 2.1 provides an overview of the JSB-RD distributed simulation environment.
- Section 2.2 describes each of the components of the JSB-RD distributed simulation environment.
- Section 2.3 provides a walkthrough of the system's operation, based on an example scenario.

2.1 JSB-RD Overview

The purpose of the JSB-RD distributed simulation environment is to provide a testbed for C4ISR research, experimentation and evaluation at AFRL/RI. Its goals include exploring the use of simulations synchronized to real-world data for the prediction of future events, as well as research into various aspects of simulation science, including:

- Adversarial modeling,
- Information management, and
- Visualization.

The JSB-RD distributed simulation environment was constructed primarily by integrating existing simulations and tools. The central component of the JSB-RD distributed simulation environment is the Joint Semi-Automated Forces (JSAF) simulation software. JSAF is a computer generated forces (CGF) system that is used by the U.S. Joint Forces Command for joint experimentation. The JSB-RD environment also includes the Ocean, Atmosphere, and Space Environmental Services (OASES) system, which models weather, and the Dynamic Terrain Simulation (DTSim), which models changes to the environment such as bomb craters, damage to buildings, and the creation and destruction of obstacles, as well as a culture/clutter simulation, which models civilian vehicle and personnel traffic. The environment also includes tools for creating scenarios from existing Air Battle Plans, extracted from the Air Operations Data Base (AODB) within the Theater Battle Management Core System (TBMCS), as well as gateways for connecting simulations communicating using the High Level Architecture (HLA) and/or the earlier Distributed Interactive Simulation (DIS) protocol, with C4ISR systems, using a variety of different mechanisms.

As noted above, the JSB-RD distributed simulation environment consists of a number of components, which are connected together using several different mechanisms. Fundamentally, the environment is an HLA federation made up of simulations that communicate using a variation of the Joint Urban Operations (JUO) Federation Object Model (FOM). Simulations that still use the DIS protocol can be connected through a DIS-HLA gateway federate. Similarly, an HLA-JBI gateway allows the federation to communicate with C4ISR system component prototypes being developed at AFRL. Finally, selected Link-16 messages can be generated using an Army software application named SIMPLE.

The overall architecture of the JSB-RD distributed simulation environment is shown in Figure 1. It consists of three primary components, addressing each of the three primary aspects of any simulation environment.

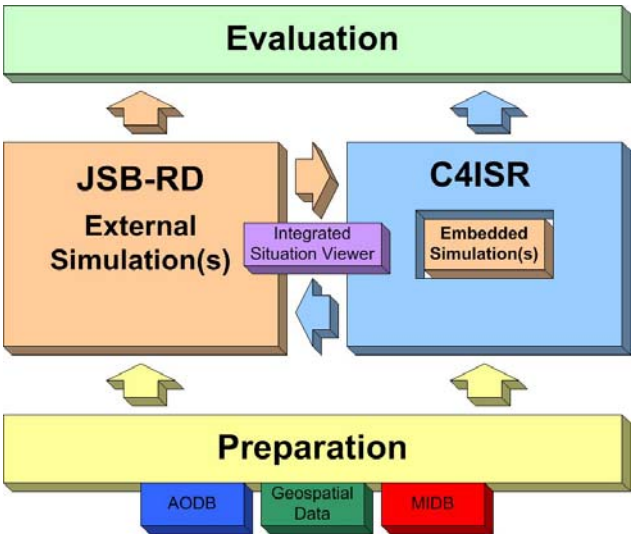

Figure 1: JSB-RD Overall Architecture

At the bottom, the Preparation component addresses experiment planning, scenario preparation, and all other aspects of preparing a simulation experiment. This includes accessing, extracting, and manipulating various kinds of source data, including Air Battle Plan (ABP) and Friendly Air Order of Battle information contained in the AODB, and Enemy Order of Battle (EOB) information contained in the Modernized Integrated Database (MIDB), as well as geospatial data. This also includes emulating the detailed mission planning that normally is performed at the unit level. In the middle are the components that address the execution phase of a simulation experiment. These consist of the various simulations that make up the JSB-RD environment, as well as real C4ISR systems, system components, or prototypes. Note that the C4ISR component can also include an embedded simulation within it, used for COA analysis or other forms of prediction. The simulation and C4ISR components communicate with one another in both directions. The states of friendly entities, and of visible opposing entities, and the effects of any relevant scenario events, are reported by the simulation to the C4ISR system, via various modeled sensor and communication systems. As the C4ISR system issues commands, they are passed back to the simulated entities that are to carry them out. In the center, visualizing the state of the simulation, as well as the plans and perceptions of the C4ISR system, either separately or together is the Integrated Situation Viewer application.

At the top, the Evaluation component provides a collection of tools for analyzing data produced by both the simulation and C4ISR systems. This component addresses the post-execution aspects of a simulation experiment. It remains the least developed part of the JSB-RD environment.

2.2 JSB-RD Components

The components of the JSB-RD distributed simulation environment are shown in Figure 2.

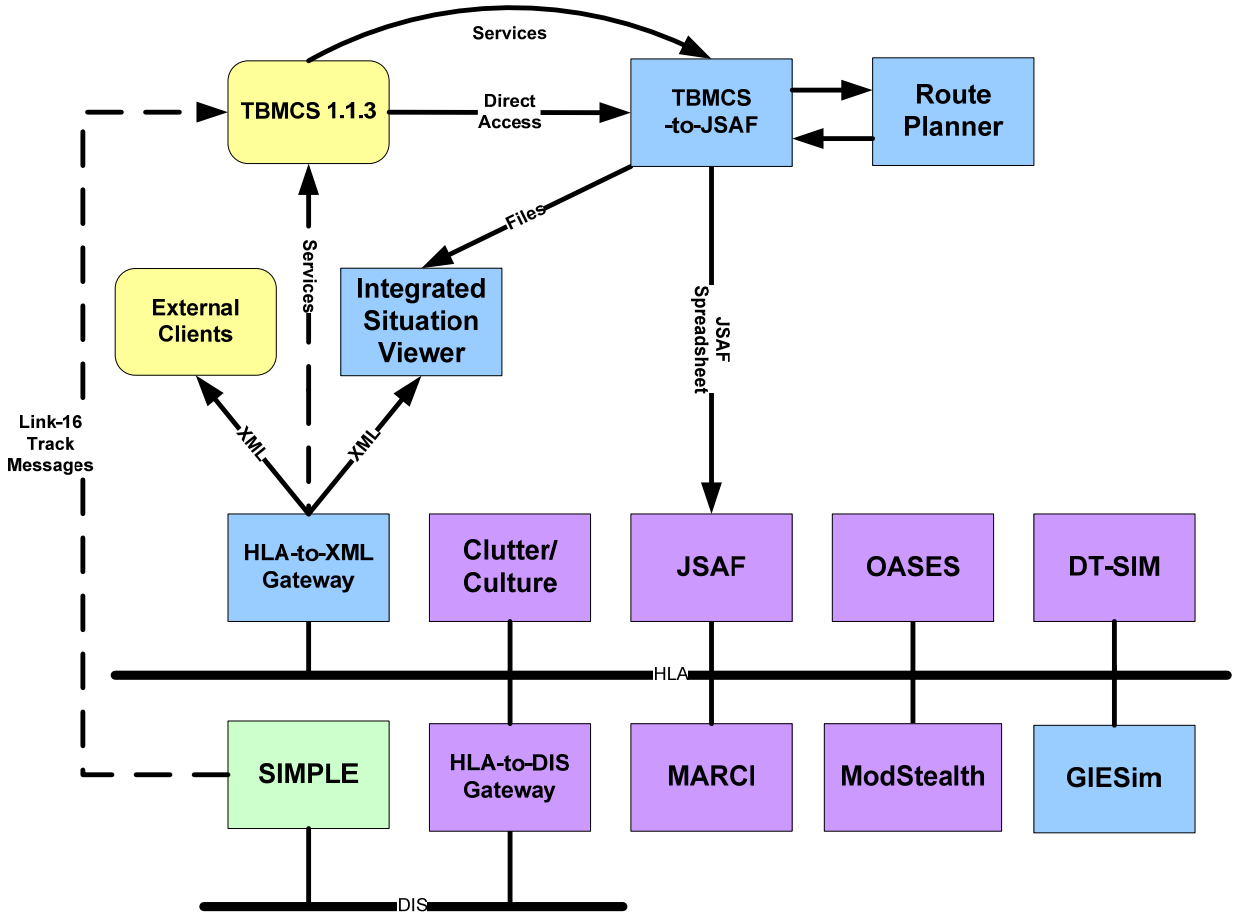

Figure 2: JSB-RD Components

JSAF, OASES, GIESim, and several supporting simulations make up an HLA federation. The HLA-to-XML Gateway translates the federation message traffic into an XML stream that can be easily read by a variety of applications. One such application is the Integrated Situation Viewer, which receives and displays entity state and interaction information. The Simulation Preparation tool extracts Air Battle Plan and Friendly Order of Battle information from the AODB, and Enemy Order of Battle information from the MIDB, which it then uses to generate a scenario input spreadsheet that can be read by JSAF to create and task the necessary simulation entities. The Simulation Preparation tool also interfaces with an aircraft route-planning tool created at AFRL to generate more realistic air mission flight paths that avoid known threats.

2.2.1 JSAF – Joint Semi-Automated Forces

JSAF is the primary simulation used within the JSB-RD environment. JSAF is an entity-level, computer generated forces (CGF) simulation system that is used by the U.S. Joint Forces Command for joint experimentation, by the U.S. Navy for Fleet Battle Experiments, and by the AFRL Human Effectiveness Directorate (AFRL/RH) in support of the Distributed Mission Training (DMT) program. JSAF provides entity-level simulation of ground, air, and naval forces. It has been used by JFCOM to simulate more than 100,000 entities within a single distributed simulation. It has also been used to support a variety of experiments with environment simulation, including dynamic terrain (i.e., craters, trenches, etc.), weather, and chemical/biological warfare defense. JSAF was originally developed by DARPA as part of its Synthetic Theater of War (STOW) program, and is descended from ModSAF. JSAF is maintained by the Joint Forces Command (JFCOM).

JSAF is an extremely large software application. It was originally developed approximately 20 years ago, in C, and has been extensively modified and extended. As a result, its original architecture has been almost completely obscured. It now consists of more than 1000 object libraries that model different types of platforms, weapons, sensors, etc. While it contains a great deal of powerful simulation functionality, it is very difficult to use, and even more difficult to modify. Documentation and support are both extremely limited.

JSAF runs under the Linux operating system. In the JSB-RD environment, JSAF can be run on multiple Linux systems simultaneously. The individual copies function together as a single HLA federate, keeping their separate internal database copies in synchronization, and sharing the computational load.

The primary input to JSAF is a spreadsheet file that defines a collection of entities, both friendly and enemy, to be created, and specifies how each entity is to be tasked. Entity types, initial locations, call signs, and assigned tasks are identified. Such spreadsheets can be prepared by hand. However, they can also be automatically generated from Air Battle Plan information, supported by Friendly and Enemy Order of Battle information, extracted from the AODB and MIDB within TBMCS.

JSAF scenarios can also be created interactively, using its integral Unit Editor. Individual entities and small units can be created, placed on the Plan View Display (PVD), and assigned tasks to perform. Interactively created scenarios can be saved and (re)loaded. However, such scenarios and the scenario spreadsheets are two separate mechanisms, and cannot be readily combined.

JSAF is capable of receiving and processing several types of information via HLA. This includes entity state information output by other simulations within the federation. For example, JSAF reads the states of civilian vehicles and pedestrians that are modeled by the clutter simulation, and displays them on the JSAF Plan View Display. Any such external entities are visible to the JSAF-controlled entities; they can be detected, fired at, collided with, etc. JSAF also reads the weather state information output by OASES although most platform and equipment models within JSAF do not make use of such information. JSAF also reads messages describing changes to the terrain that are output by DTSim. Such dynamic terrain changes (e.g., the appearance of a bomb crater) can affect the movement of ground vehicles in JSAF.

The primary output of JSAF is entity state information for all of the entities that it models. It also outputs various types of interactions, including weapons fire and detonation, collision, etc. In support of various exercises and experiments, JSAF has been extensively modified to support additional FOMs. It includes an extensive FOM agility layer. Many of its outputs are platform- or weapon-system specific.

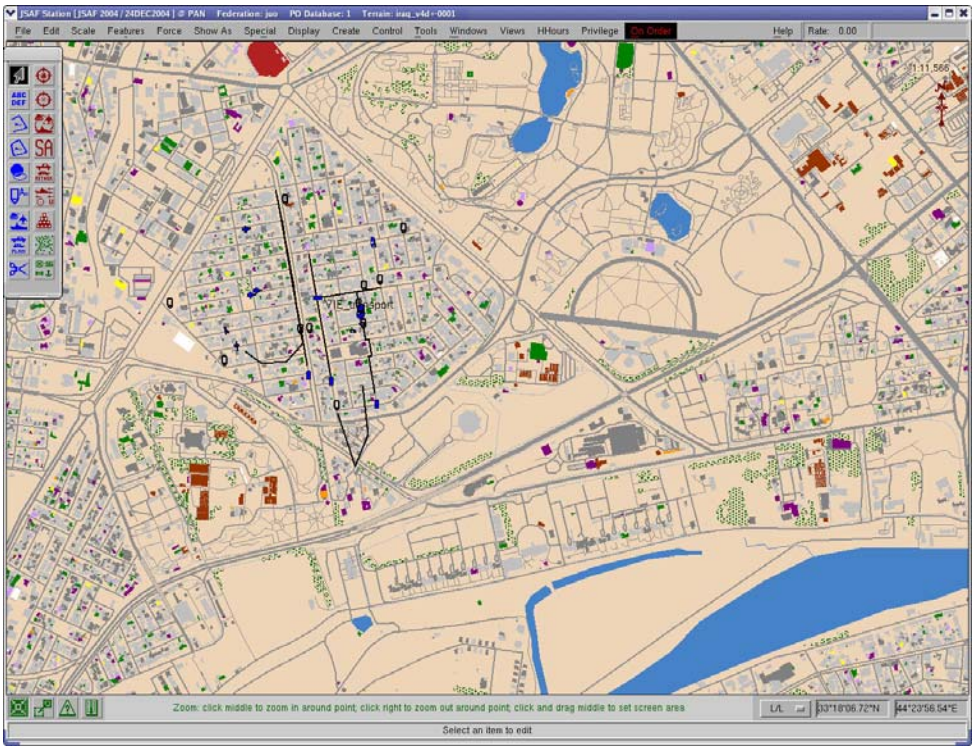

Figure 3: JSAF Plan View Display with Air Dynamics Editor

Figure 3 shows the JSAF PVD, with a map background (showing part of Baghdad) and platform-level icons. The toolbar at the upper left provides access to a variety of tools for creating, tasking, and otherwise manipulating the simulated entities. Multiple copies of the JSAF PVD can be run simultaneously. Commonly, some JSAF GUIs are congifured as controller workstations, which see and can manipulate all entities, while others are configured as “player” workstations, which can see only their own forces, as well as any enemy forces that have been detected.

The JSB-RD simulation environment currently uses a version of the Joint Urban Operations (JUO) federation object model (FOM). The JUO FOM was developed by JFCOM for use in the Joint Urban Resolve exercise. It is based on the Realtime Platform Reference (RPR) FOM, which was designed to map the original DIS protocols into an HLA environment. The JUO FOM includes a number of objects and interactions that were added to support specific aspects of the Millenium Challenge 2002 (MC02) and Joint Urban Resolve exercises. Most of these are not currently used within the JSB-RD environment. The JUO FOM elements that are currently used include:

- the BaseEntity/PhysicalEntity/Platform class, primarily the GroundVehicle and Aircraft subclasses,
- the Atmosphere, SurfaceWeather, and Weather classes, and their various subclasses,
- the Collision, Weapon Fire, and Munition Detonation interactions, and
- the GIESim Entity State, Message Send, and Message Received interactions.

While developing JSAF scenarios, aircraft/ordnance/mission combinations that don't appear to work properly together were encountered frequently enough to warrant compiling a table of combinations of interest and evaluating their functionality.

A list of aircraft and weapon types from the "Air Force 2015 Plan" that are supported by JSAF 2004 was compiled, and cross-indexed with a list of JSAF-supported mission types from the "Force Level Operational Mission Models, Rev 5" (FLOMMR) document. Each combination was flown under controlled conditions and rated "successful," "broken," or "conditional." Any combination rated as other than "successful" was re-flown under various conditions and the rating adjusted as appropriate.

Of primary interest was how the aircraft entities in question interacted with other entities during a simulation – especially combat interactions. With that interest in mind mission types were first divided into two main categories: combat and non-combat. The combat category was further divided into air-to-ground missions and air-to-air missions. In the non-combat category aerial refueling was of special interest, while the rest of such missions were grouped as "other."

Air-to-ground missions: the FLOMMR defines two forms of ground attack, "Direct Attack" and "Standoff Attack." JSAF has no such mission categories, providing rather (with some few exceptions) mission types of "Interdiction," "Strike," and "Target of Opportunity." There seemed to be little difference between Interdiction and Strike missions in JSAF, although aircraft executing the Strike mission generally seemed to launch ordnance at greater ranges than while executing the Interdiction mission with the same weapon.

For the purpose of this evaluation a mission was considered to have been flown successfully if an aircraft/ordnance pairing successfully delivered the ordnance (regardless of whether a hit was scored) against first a T-72 tank, and then (if the aircraft failed to launch against the T-72) against an SA-10 launcher. A "conditional" result was scored if a mission was successful against one type of target while unsuccessful against the other. In all cases the attacking aircraft was given a lengthy, straight approach to the target using mission defaults (as defined by JSAF) for speed and altitude. Before either a "conditional" or "broken" result was decided a mission was re-flown multiple times, using varying approach distances and altitudes, in an attempt to mitigate possible limitations in a weapon's intended delivery envelope.

Air-to-air missions: The related FLOMMR categories are "Escort" and "Station." In JSAF terms, multiple mission types (often aircraft-dependent) are defined that relate to (direct or potential) air-to-air combat as their focus. JSAF provides one direct mission match in the form of "Escort Designated Aircraft," as well as "Station type" missions in the form of "DCA/OCA" and "TCP CAP." It provides a direct combat-intent mission, "Intercept/Attack Designated Aircraft," and the more generic "Target of Opportunity" mission. In a more general sense, almost any mission should result in the aircraft launching on a perceived airborne target within weapons range if the aircraft's weapons permission is set to "Free" manually by the operator after the mission is assigned.

For the purpose of this evaluation a mission was considered to have been flown successfully if an aircraft attacked a provided target aircraft with available weaponry, regardless of whether the attack actually destroyed the target. If an aircraft failed to launch against a target the mission was re-flown multiple times at various ranges and altitudes in an attempt to mitigate possible limitations in a weapon's intended delivery envelope.

Tanker missions: This mission type was evaluated both from the perspective of the tanker aircraft itself and from that of the combat aircraft taking on fuel. For a tanker to fly a successful mission it had to assume a suitable orbit and deliver fuel when requested. For a combat aircraft to fly a successful mission it was required to rendezvous with a designated tanker and then to actually take on fuel.

Other missions: The other types of FLOMMR missions either did not map well to JSAF mission types, or did not entail interactions of the type that could be represented by JSAF running as a stand-alone simulation. For that reason they were not evaluated.

The results are summarized in Table 1. There was a consistent failure of aircraft to employ guns or cannons in attacking targets, even when guns were specifically enabled while assigning a mission. This failure resulted in aircraft with all-cannon armament, such as the AC-130H Gunship, to be ineffective in any assigned combat mission.

During successful combat missions the aircraft did not need to be manually set to weapons free mode. That mode would automatically change to "Free" at the appropriate time, then revert to "Hold" after weapons release.

Nearly all air-to-ground missions were eventually completed successfully. There were some unusual behaviors observed, which are described in the spreadsheet "Notes" column. In most cases where entities were not initially attacked, it was possible to correct the behavior by zooming in very closely to the target(s) and placing the attack point precisely beneath a target vehicle. This requirement seemed to be munitions-dependent (most notable with respect to the GBU10 bomb), as in many cases it was sufficient to place the target marker over the area of a target vehicle group while zoomed out to a wider-scale view. In the case of a mission where attacking was a reactive behavior, such as the "Target of Opportunity" mission, the aircraft became confused by multiple task frame switching and never reacted to the target vehicle. In one special case, the UAV Predator executing a "Predator Strike" mission, the ordnance launch was ruled unsuccessful because the missile consistently flew well past the target as though never properly aimed. This is distinct from a "miss" where the ordnance might explode in the vicinity of the target but fail to do damage.

The most notable result of the air-to-air combat missions was the complete failure of the "Intercept and Attack" mission. In every case the attacking aircraft would successfully intercept the target aircraft, but would then simply follow along in close proximity without attacking. In a similar vein, aircraft given the "Escort" mission would fly in close formation with the escorted aircraft, but would fail to engage any attacking aircraft, regardless of whether it or its charge were attacked. In cases where weapons permission was manually set to Free weapons would usually (but not always) be launched if the attacking aircraft happened to approach within weapons envelope of the escorting aircraft (such as a nose-to-nose intercept), but the escort never executed any actual attack reaction. Two very successful reactive missions were the "DCA/OCA" and "Target of Opportunity" missions. In each case the aircraft would switch to attack mode as it perceived a hostile aircraft, and then successfully engage with missile fire. The "TCP CAP" mission was of marginal utility as the aircraft had to be set manually to weapons free and a target would have to happen across its field of fire before a weapon would be launched.

Aerial refueling generally worked well—with one caveat. In the JSAF implementation the refueling aircraft must remain connected to the tanker for the entire duration of the process, and then all of the fuel is transferred in a single instant at disconnect time. This is obviously a design decision, with the unfortunate result that if an aircraft were to be called away from refueling after expending, say, half of the normal time it would not benefit from receiving 50 percent of a normal fuel load.

For the most part the missions detailed in this study can be successfully executed in JSAF. In the cases where a given aircraft/munitions/mission combination seems to be non-functional, a work-around or technique has been described which will result in success; in the cases where no such remedy was discovered those combinations have been documented so that they can be avoided.

Table 1: JSAF Aircraft/Munition/Mission Test Results

Aircraft	Munition(s)	Interdiction	Strike	Tgt of Opp	Refuel	DCA/OCA	Intercept	TPT CAP	Escort	Orbit
A/OA-10A Thunderbolt II	AGM65 Maverick	success	success	success	broken	N/A	N/A	N/A	N/A	Note 3
AC-130H Gunship	M50 20mm	broken	broken	N/A	N/A	N/A	N/A	N/A	N/A	broken
AC-130U Gunship	PGU22 25mm	?	?	?	N/A	N/A	N/A	N/A	N/A	?
	PFHE 40mm	?	?	?	N/A	N/A	N/A	N/A	N/A	?
	M546 105mm	success	success	success	N/A	N/A	N/A	N/A	N/A	Note 3
Airborne Laser (AL-1)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
B-1 Bomber	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	AGM154B (JSOW-B)	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	CBU103 CEM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
B-2 Spirit	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
B-52H Stratofortress	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	Mk82 500lb Bomb	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
C-130H Hercules	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
C-130 Hercules	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
C-17 Globemaster III	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
C-5 Galaxy	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CV-22 Osprey	N/A	N/A	N/A	N/A	success	N/A	N/A	N/A	N/A	N/A
E-3 AWACS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
E-8 JSTARS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
EC-130E ABCCC	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
EC-130H Compass Call	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
F-117A Nighthawk	GBU10 2000lb Bomb	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
F-15C Eagle	Air-to-air missiles	N/A	N/A	Note 1	success	Note 1	Note 2	Note 3	Note 4	Note 3
F-15E Strike Eagle	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	GBU10 2000lb Bomb	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	CBU105 SFW	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	AGM65 Maverick	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	AGM88 Harm	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	Air-to-air missiles	N/A	N/A	Note 1	success	Note 1	Note 2	Note 3	Note 4	Note 3
F-16C Fighting Falcon	AGM65 Maverick	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	CBU105 SFW	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	CBU104 GATOR	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	GBU10 2000lb Bomb	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	Air-to-air missiles	N/A	N/A	Note 1	success	Note 1	Note 2	Note 3	Note 4	Note 3
F-16CJ Fighting Falcon	AGM88 Harm	broken	broken	N/A	N/A	N/A	N/A	N/A	N/A	broken
F-35A Joint Strike Fighter	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	Air-to-air missiles	N/A	N/A	Note 1	success	Note 1	Note 2	Note 3	Note 4	Note 3
F/A-22 Raptor	JDAM	success	success	success	success	N/A	N/A	N/A	N/A	Note 3
	Air-to-air missiles	N/A	N/A	Note 1	success	Note 1	Note 2	Note 3	Note 4	Note 3
HH-60X	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
KC-10A Extender	N/A	N/A	N/A	N/A	orbits OK	N/A	N/A	N/A	N/A	N/A
KC-135 Stratotanker	N/A	N/A	N/A	N/A	orbits OK	N/A	N/A	N/A	N/A	N/A
MC-130P Combat Shadow	N/A	N/A	N/A	N/A	broken	N/A	N/A	N/A	N/A	N/A
RC-135 Rivet Joint	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
UAV Global Hawk (RQ-4)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
J-UCAS	FGB	Note 5	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Note 5
	FSWC	success	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	FSWS	broken	N/A	N/A	N/A	N/A	N/A	N/A	N/A	broken
	SSB	Note 5	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Note 5
	FLAW Pod	Note 5	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Note 5
	NCAS	Note 5	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Note 5
UAV Predator	AGM114 Hellfire	N/A	broken	N/A	N/A	N/A	N/A	N/A	N/A	broken

Notes

1 success; switches to attack reaction and engages; guns don't work
2 intercepts target but doesn't attack
3 will launch on targets that come into range/view if permission manually set weapons free
4 forms on escortee, but does not react to viable enemy targets/attackers
5 insists on trying to use FSWC, doesn't try specified munition

The advantages of using JSAF as the central component of the JSB-RD distributed simulation environment include:

- It supports a very wide variety of entity types, including ground forces, aircraft, ships, satellites, lifeforms.
- Entities include components such as sensors, comm, weapons, decision logic, etc.
- It supports complete units, as well as individual entities, including communication networks.
- It provides a large collection of automated tasks and behaviors at both the unit and entity level, using finite state models.
- It provides three different scenario creation mechanisms:
  - Plan View Display
  - Spreadsheet Mechanism
  - Dynamic Retasking – via the Viewer

However, JSAF also has significant disadvantages, which include:

- JSAF consists of a very large library of very old code written in C with some C++; it is extremely difficult to understand, modify or extend.
- Many of the behaviors provided by JSAF are incomplete or broken; JFCOM maintains JSAF only with respect to its current exercise objectives, rather than as a general-purpose capability. Behaviors and other functionality that are not immediately needed are often damaged in the course of meeting short-term objectives.
- Considerable trial and error is often involved in scenario development

2.2.2 OASES

The OASES system is a suite of applications for creating and managing a three-dimensional, time-varying, digital representation of the natural environment. OASES has been used primarily to provide synthetic natural environments (SNEs) to systems of networked military training simulations running on the Department of Defense's (DoD) High Level Architecture (HLA). The simulated natural environments created by OASES are based on authoritative, validated numerical models, typically the same models that are used by METeorological/OCeanographic (METOC) personnel in support of real-world military operations. OASES provides tools for converting authoritative model outputs to a data format recognized by all of the OASES applications. This format supports the data access requirements of distributed simulations that integrate virtual and/or live entities and which must operate in real-time. Additionally, OASES provides tools for tailoring the SNE, either before the simulation begins or while it is running, to meet exercise-specific requirements for environmental phenomena.

The original development of the OASES system was sponsored by the Defense Advanced Research Project Agency (DARPA), under the name *TAOS* (Total Atmosphere Ocean Services), in support of the Synthetic Theater of War (STOW) 1997 Program. In 1998, DARPA funded the development of a low-resolution worldwide atmospheric and oceanographic database, also known as the *Global-98* database, for use by the JSIMS program. In 1999, the United States Space Command's (USSPACECOM) Space Warfare Center funded extensions to TAOS to support the space environment, specifically ionospheric effects on precision-guided missiles, as part of the PSM+ (extended Portable Space Model) project. More recently, funding for continued development and integration with the HLA, under the name OASES, has been provided primarily through the Environment Federation (EnviroFed) projects.

The OASES system consists of five primary subsystems. The OASES Ingestor converts all input model data to a common run-time format that is recognized by all OASES subsystems. The OASES Transformer uses a set of transformation algorithms to augment existing OASES databases with various environmental parameters that are not provided directly by an external data source, but that are required by the simulations served by OASES. The OASES Editor allows users to tailor the contents of an OASES database. The Editor provides three editing algorithms: 1) replacement at a point with gaussian spatial and temporal blending, 2) a Pressure Field Modification (PFM) algorithm for editing atmospheric environments while preserving correlation between temperature, pressure, wind and relative humidity, and 3) a precipitation editing algorithm. The Editor can be used to prepare scripted changes to an existing METOC scenario, or it can be used at run-time to modify the SNE during a simulation exercise. The OASES Time Federate and Publisher are the subsystems that interface directly to the HLA Federation; that is, they are the OASES *Federates*. They create and update the objects that encapsulate the state of the simulated natural environment via services provided by the RTI. The Time Federate creates objects that establish the time-dependence of the SNE while the Publisher manages objects that encapsulate its spatial-dependence.

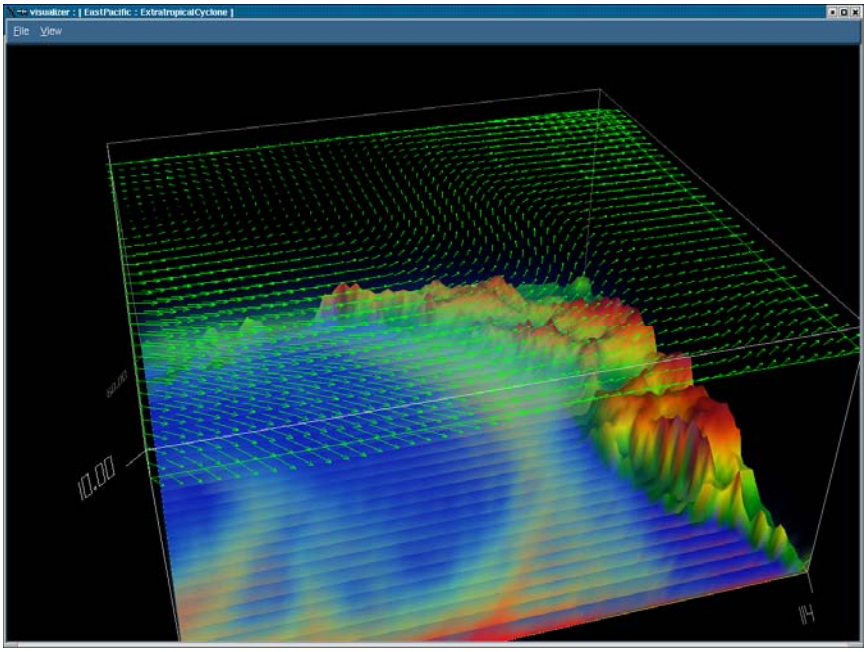

Figure 4: OASES Weather Visualization

The OASES Visualizer is a tool for visualizing the contents of an OASES database. It is used to validate databases built by the Ingestor and/or extended by the Transformer, to review the results of edits applied by the OASES Editor, to monitor the current state of the SNE created by the Publisher, and/or to monitor the current state of the SNE as received by the OASES Subscriber. An example of an OASES Visualizer display is shown in Figure 4.

Finally, the OASES Receiver is responsible for polling local or remote data sources, using the Internet File Transfer Protocol (FTP), for environmental data transmittals matching a user-specified file-naming pattern. The Receiver is the subsystem that supports the“Live Mode” of OASES, in which the simulated natural environment is continuously updated based on the data received from current and forecast environmental models running in real-time.

Within the JSB-RD environment, OASES is used to bring in weather information from Air Force and Navy sources. OASES outputs weather state information, including temperature, pressure, and precipitation information, in one-dimensional (profile), two-dimensional (surface), and three-dimensional forms, over HLA. This information is read by JSAF and DTSim, which use it to modify some military operations, and to implement changes to the terrain database, respectively.

2.2.3 Culture/Clutter Simulation

The Culture/Clutter simulation, which is part of the JSAF distribution, models the movements of civilian vehicles and pedestrians. The amount and type of clutter is specified using clutter templates. Each template specifies a list of entity types with associated relative weights, as well as a collection of control points. Each control point specifies a center location, a radius (defining a circular area), and a number of clutter entities. Each control point is identified as defining a

static clutter area, a mobile clutter area, a clutter source, or a clutter sink. Clutter entities move randomly with a static clutter area. Sources and sinks allow dynamic traffic flows to be created. Clutter etities are randomly created in the source areas, and move to random locations within a sink area. When they arrive, they are destroyed and replaced with a new entity in one of the source areas. The clutter simulation publishes entity state information for each of the clutter entities as they move.

In support of the JFCOM Urban Resolve joint experiment, the Culture/Clutter simulation was significantly enhanced to support a wide variety of clutter entities, including both vehicles and lifeforms. Templates can be defined that specify the movements of clutter entities at specific times. These templates can be used to more realistically model civilian traffic movements such as commuting to and from work, political demonstrations, etc.

2.2.4 DTSim

The Dynamic Terrain Simulation (DTSim) models changes to the terrain component of the environment. The changes result from various types of simulation events, including weapon detonations, movement, weather effects, and military engineering operations, such as the creation and destruction of obstacles. DTSim receives interaction events from JSAF, and determines what effect, if any, they have on the geometry or attributes of the terrain at the location event. For example, when DTSim receives a weapon detonation interaction from JSAF, it may, depending on the type of the weapon and its proximity to the terrain surface, or a specific terrain feature, determine that a crater should be created, or that a building should be damaged. DTSim updates its internal terrain representation, and publishes messages over HLA describing how the terrain has changed. These messages are used by JSAF to update its terrain representation, which in turn affects how some activities are carried out. For example, the appearance of a new crater may change the movement of vehicles to avoid it. Weather effects, such as prolonged rain, may also change the characteristics of the terrain, restricting the speed of cross-country movement.

2.2.5 ModStealth

ModStealth is a 3D “stealth” viewer that is part of the JSAF software distribution. It provides three-dimensional perspective views of the scenario entities and environment that are dynamically updated as entities move and events occur. Figure 5 shows an example perspective view of part of the Baghdad urban database. The viewpoint can either be attached to a specified scenario entity, or can be manually manipulated. The ModStealth control panel, which is used to manipulate the viewpoint, including attaching the viewpoint to a specific entity, is shown in Figure 6.

The use of ModStealth to capture screenshots of scenarios exposed several performance issues. In particular, ModStealth requires a specialized database format that is different from the CTDB format used by JSAF. The extremely large and detailed Baghdad database significantly stressed ModStealth’s memory management capabilities.

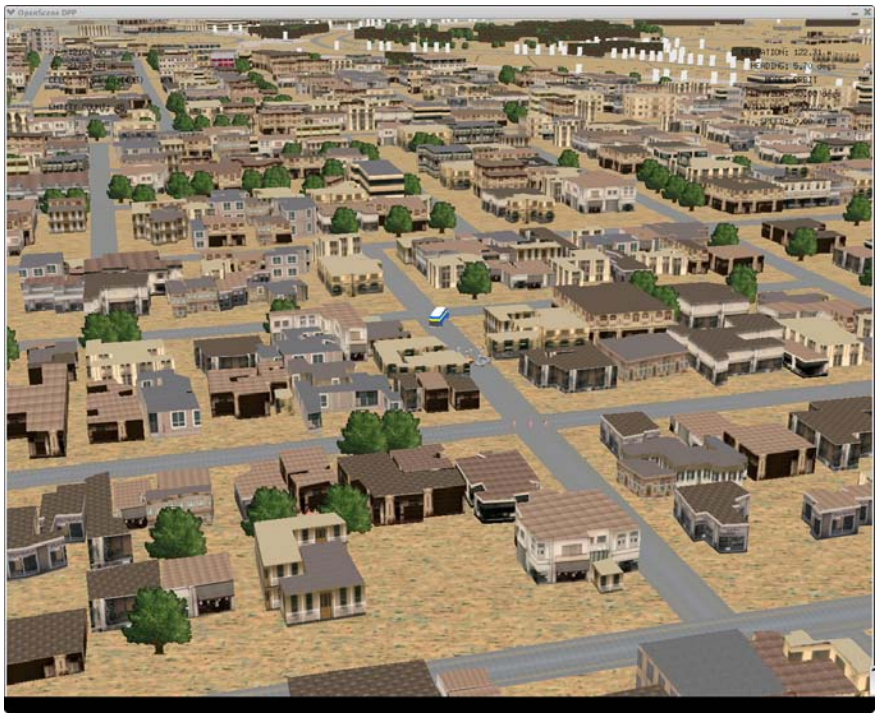

Figure 5: ModStealh Perspective View

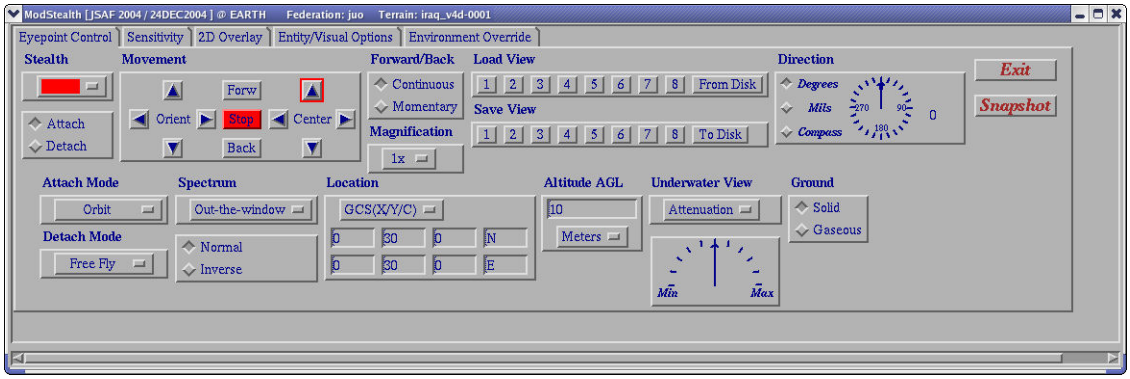

Figure 6: ModStealth Control Panel

2.2.6 MARCI

MARCI (Multi-host Automation Remote Control and Instrumentation) is a simulation exercise control and management tool that is part of the JSAF software distribution. MARCI allows an operator to control, monitor, and analyze an entire federation, possibly distributed across multiple sites, from a single workstation. MARCI can be used to distribute federates to multiple systems prior to a simulation run. MARCI's Mass Launch capability provides the ability to start multiple systems nearly instantaneously. MARCI can also display disk space and memory utilization on these systems. Operators have the ability to launch individual workstations by choosing from a list of common options in a graphical user interface (GUI). Individual federates can be started and shut down. MARCI can execute federation-wide Global Pause and Global Resume operations through the RTI. It can execute federation-wide scenario load and save operations.

2.2.7 HLA-to-DIS Gateway

The HLA-to-DIS Gateway is part of the JSAF software distribution. It is an HLA federate that translates a subset of the MC02 FOM messages, the subset that matches the Realtime Platform Reference (RPR) FOM, to and from corresponding DIS Protocol Data Units (PDUs). This allows DIS applications to participate in HLA federations.

2.2.8 SIMPLE

SIMPLE (Simulation to C4I Interchange Module for Plans Logistics and Exercises) is an interface between the simulated battlefield environment and real world command and control systems. SIMPLE provides a database that maps simulation units, platforms, munitions, and supplies to real world units, platforms, munitions and supplies.

SIMPLE also contains a messaging module that correctly generates the tactical messages required by the military C4I systems to report on these units, platforms, etc.

The heart of SIMPLE is the scenario database. This database is uniquely tailored to each simulation scenario in order to provide the correct mappings from "sim" to reality. SIMPLE is a product in the Digital Battlestaff Sustainment Training (DBST) federation environment developed primarily by the National Simulation Center (NSC) located at Ft. Leavenworth, KS.

SIMPLE was used to generate air track messages in Link-16 format, for input to the TBMCS Track Management Data Base (TMDB).

2.2.9 HLA-to-XML Gateway

The HLA-to-XML Gateway is an HLA federate that translates FOM entity and interaction messages into an easily parsable XML stream that can be accessed by multiple applications. The purpose of the HLA-to-XML Gateway is to make it easier for a variety of applications to access the data generated by an HLA federation, by encapsulating the details of connecting to an HLA federation and receiving data. The HLA-to-XML Gateway is primarily used to support the Integrated Situation Viewer.

2.2.10 Integrated Situation Viewer

The Integrated Situation Viewer, commonly referred to simply as “the Viewer”, is intended to support experimentation with theater-level air mission situation awareness and dynamic retasking. The Viewer displays simulation state information describing air missions, which is, in effect, ground truth, and the corresponding Air Battle Plan information. In addition, any information (derived from the simulation) that reflects the reported or perceived current situation, as output by various sensor models, is displayed.

The Viewer consists of a number of loosely coupled components. The back-end portion of the Viewer consists of a collection of components that are capable of reading data from various sources. Simulation entity state and event data output by JSAF, OASES, DTSim, and other simulations is read via an HLA-to-XML gateway. This gateway converts entity state information received over HLA into a stream of XML messages. It also performs coordinate conversion (from geocentric coordinates to geodetic coordinates), and partial translation of the DIS Entity Bit Vector (EBV) fields that are used to hierarchically identify entities by nationality, domain, type, subtype, etc. Other data sources, including JSAF input spreadsheets, are read directly from the appropriate files.

The “heart” of the Viewer consists of an integrated domain model that stores and maintains information on air mission plans, individual aircraft, and their targets. The data read from the various back-end sources is used to populate and update this model.

The front-end of the Viewer consists of multiple views of the information that the domain model contains. Several types of views are supported, including:

- Map views – displaying the locations of aircraft and targets, planned and actual flight paths, etc., overlaid on a map background at multiple scales and resolutions using the JView visualization toolkit developed by AFRL/RISB,
- Tabluar views – listing entities of various types (aircraft, targets, missions, etc.) and their relevant characteristics, and capable of being sorted and ordered in various ways,
- Text views – displaying a streaming list of text messages, generated by the Simulation Network News (SNN) utility, which is part of the JSAF distribution.

The Viewer also includes an interactive retasking capability. When this capability is invoked, it sends messages to JSAF to alter the current tasking of a specified aircraft, normally designating a new target for that aircraft.

The main Viewer GUI, shown in Figure 7, consists of a default Map View and a row of buttons at the bottom of the window to start other views. With the exception of the SNN View, multiple instances of all of the views can be launched. The Map View contains entities, routes and background map data. It is also where the lens appears.

Figure 7: Map View GUI, Including Lens

The lens feature allows the user to left click anywhere in the main GUI, causing both a lens and a separate magnified Map View to appear. The lens can be moved by clicking and holding the mouse button within the title bar region of the lens window. Clicking in another location on the map will cause the lens to snap to that location. The movement of the lens is reflected by the map information in the Magnified Map View.

The Magnified Map View, shown in Figure 8, shows the same data as the Map View but with both the CADRG and DTED map background at a higher resolution. The center divider can be moved to adjust the amount of space both map types take up on the screen. Any entity icon selections made in the Magnified Map View will result in the selected item being highlighted in the Table View if it is visible there. Route selections made in the Magnified Map View will result in the Route Name being output to the command window.

The Table View is launched via the New Table View button at the bottom of the main GUI window. This view contains data related to the entities shown on the map. The columns can be moved around and the rows can be sorted by clicking on the column header. Complex sorts are enabled by holding the control key down when clicking subsequent column headers. A row selected in the table causes the related entity icon to be highlighted on the Map View.

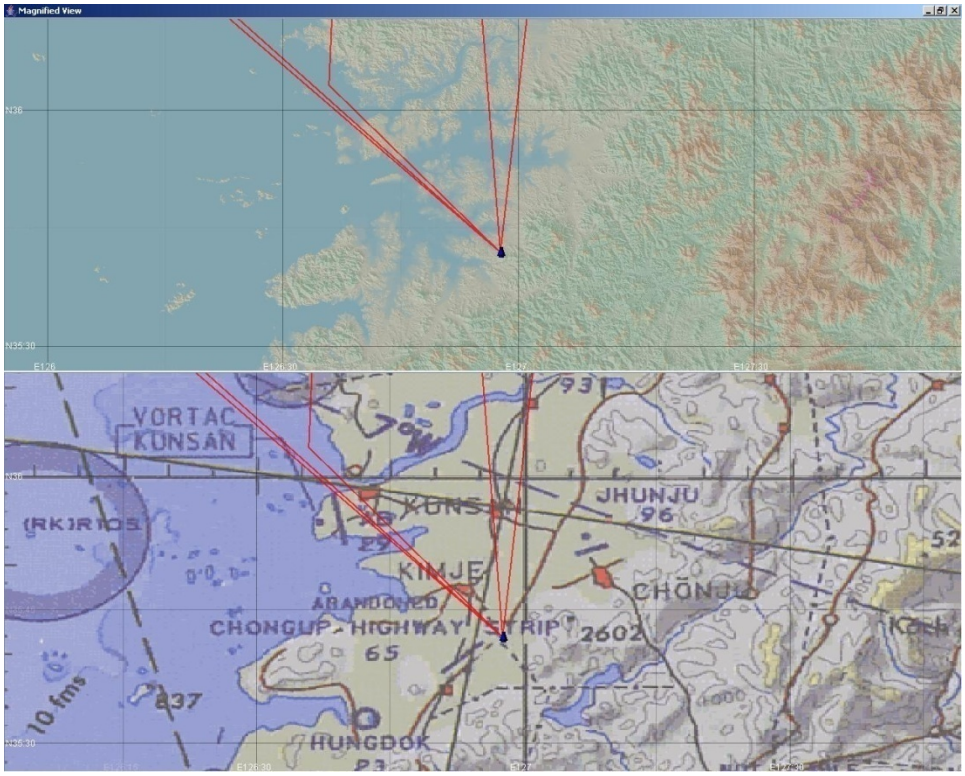

Figure 8: Magnified Map View

The Controller is the "brain" of the Viewer, coordinating all of the other components. When new information arrives from one of the sources, the Controller triggers the updating of the domain model, and then the updating of all views that are affected by the change. Similarly, when the interactively changes a piece of information in one of the views, the Controller triggers the updating of the domain model, and then the updating of any other views that are affected by the change. The Controller also coordinates the selection of entities and locations across all of the views, so that an entity that is selected in one of the views is also highlighted in all other views in which it appears. A Controller GUI allows the user to select which types of views should be displayed and what the content of each should include.

2.2.11 TBMCS-to-JSAF

The simulation preparation component of the JSB-RD environment, TBMCS-to-JSAF, allows existing Air Battle Plans (ABPs) contained within the Air Operations Data Base (AODB) of the Theater Battle Management Core System (TBMCS) to be converted into sets of JSAF input spreadsheets that can be executed using JSAF. This application extracts a specified ABP from the AODB, which contains specifications of multiple air missions of various types. Each mission specification includes the numbers and types of aircraft involved in the mission, their takeoff and return times and bases, and a sequence of key mission events. These mission events include takeoff, refueling (start and end), time on target (start and end), and landing. Ground attack missions also identify their respective targets. Supporting information describing the mission targets is extracted from the MIDB.

The basic mission information, along with a list of the air defense threats in the area, can be fed to a separate Route Planner application (see below), which determines the "best" route for each mission to and from its assigned target while avoiding air defense threats. The returned route contains a number of intermediate waypoints that the aircraft should pass through on the way to and from their target.

This information is then used to generate a JSAF input spreadsheet. The spreadsheet contains two entries for each scheduled air mission, one describing the ingressing leg of the mission, and the other describing the return leg. Each entry specifies, in JSAF terms, the number and type of aircraft, the call sign(s) of the aircraft, the type of task to be performed, the take off and return times, and the base and target locations. A second spreadsheet contains the intermediate route points, which are linked to the missions by name.

2.2.12 Route Planner

The Route Planner application takes a "stick route" for a planned air mission, consisting only of the source airbase coordinates, the target coordinates, and the return airbase coordinates, as well as a collection of adversary air defense threats (i.e., SAM and AAA sites). It returns a more complex route, containing additional waypoints, that attempts to reach the target and return while avoiding the listed threats.

The Route Planner is used by TBMCS-to-JSAF to attempt to emulate the more detailed mission planning activities that occur at the unit level. It passes the Route Planner the basic mission information obtained from an Air Battle Plan, and the air defense threats obtained from the MIDB. It takes the resulting route, with the added waypoints, and constructs a spreadsheet that can be read by JSAF.

2.2.13 GIESim

The Global Information Enterprise Simulation (GIESim) provides high fidelity Link-16 network modeling with full resolution of propagation effects, including power and distance based Signal to Noise ratio, terrain masking and other Line Of Sight (LOS) issues. The vision of GIESim is to move, process, manage, and protect the C4ISR information that supports the functions of Global Awareness and Dynamic Planning and Execution. The mission of GIE is to link aerospace assets in-theater and globally, to integrate C3 & ISR networks, to defend critical information systems from cyber attack, and to develop new information processing and management techniques. Most large-scale force level simulations assume perfect communications. The lack of communications in a simulation environment can lead to the prediction of erroneous results.

Within the GIESim framework, users are able to execute, via a common interface, multiple communications and network M&S tools to effectively and efficiently analyze candidate communications architectures and technologies. GIESim can interface with other M&S tools (e.g., force-level simulations and detailed hardware system models) to provide the appropriate level of M&S fidelity and processing speed for the broad spectrum of M&S tasks.

Within the JSB-RD distributed simulation environment, the role of GIESim is to evaluate communications connectivity between various entities in the simulation. Communication networks are defined within GIESim, tying together various collections of simulated entities.

JSAF controls the movements of the simulated entities. GIESim monitors customized entity state messages published by JSAF and updates the locations and headings of the entities. When two entities need to communicate with each other, JSAF sends a request to GIESim identifying the two entities, the type of communication, and the length of the message. GIESim then determines whether or not the specified entities can communicate, either directly or via available relays. If they can communicate, GIESim returns a response to JSAF indicating the delay before the message will arrive at its destination. JSAF then schedules the delivery of the message at the indicated time.

2.2.14 Logging and Analysis

Under this effort, additional logging and analysis tools were added to the JSB-RD distributed simulation environment. These tools included:

- JLogger – an open source, general-purpose data logging tool,
- CDLA – logger, playback, debug, network traffic tool
- FAARS – after-action review tool

JLogger is a general-purpose data logging tool for Java developers. It is used in conjunction with JSAF to log all HLA object and interaction messages that are output during a simulation run. The logged messages are captured and stored in a relational database management system.

CDLA (CWIN Data Logging and Analysis) is a tool developed within Northrop Grumman's Cyber Warfare Integration Network (CWIN) to provide a data logging, playback, debugging, and network traffic analysis tool. It was obtained and added to the JSB-RD distributed simulation environment under this effort.

FAARS (Future After Action Review System) is an analysis tool that can be used to analyze the results of HLA simulations. It was developed by the Army, and is particularly well-suited for analyzing ground combat scenarios.

2.3 Example Scenario

This section describes one of the scenarios developed and executed using the JSB-RD during this effort. The scenario uses the unclassified Pacifica MIDB database, in which the state of California is redefined to be the adversary nation of Califon. The scenario consists of two parts. First, there is a theater-level air superiority scenario, with US air forces having the operational-level objective of disabling the Califon Integrated Air Defense System (IADS). In addition, there is a small-scale scenario involving a small group of insurgents attacking a US convoy using a truck bomb. The scenario also incorporates elements of the Empire Challenge 2008 airborne networking experiments.

2.3.1 Scenario Development

The scenario includes a total of 263 separate entities. The adversary forces include:

- SAM sites of several different types (SA-10, SA-5, SA-3, and SA-2), with the associated Regiment & Brigade HQs, all with their full complements of radars and communications,
- MiG 29s, with their search radars.

The friendly forces include:

- Strike aircraft, including B52s, F117s, F15Es, F16Cs, F18EFs – attacking the Califon IADS,
- C4ISR aircraft, including AWACS, E6B, E2C, Global Hawk, Predator, C9, Boeing 767s,
- Ground facilities, including a Joint Air Operations Center (AOC), a Cyber AOC, a Space AOC, a Wing Operations Center (WOC), an Air Support Operations Center (ASOC), and a Combat Reporting Center (CRC),
- A collection of Unattended Ground Sensors (UGS),
- A US Army truck convoy,
- A US Special Operations Forces (SOF) squad, with an associated Joint Tactical Air Controller (JTAC).

In addition, the scenario includes some civilian ground vehicle traffic, a group of five insurgents, and a truck which they have loaded with explosives.

ID	Function	Mission #	Callsign	HLA Class	HLA Entity ID	FOM Entity ID	GIESIM ID	JSAFType	JSAF ID
175	ALAMITOS SA-10 C2 BNKR		B0072_01_98	66	435	1043		Bunker	JS/80842635/14e7/1446
176	ALAMITOS SA-10 C3 VEH		B0072_01_04	67	420	1003		SA-10 Command Vehicle	JS/80842635/14e7/1440
177	ALAMITOS SA-10 CLAM SHELL RDR		B0072_01_07	66	441	1050		Clam Shell Radar	JS/80842635/14e7/1448
178	ALAMITOS SA-10 FLAP LID RDR		B0072_01_02	67	432	1027		SA-10 FCR	JS/80842635/14e7/1444
179	ALAMITOS SA-10 LNCHR01		B0072_01_00	66	424	1015		SA-10 TEL	JS/80842635/14e7/1442
180	ALAMITOS SA-10 LNCHR02		B0072_01_01	66	426	1021		SA-10 TEL	JS/80842635/14e7/1443
195	AWACS Airspace Control		CHALICE	64	535	1432	301	E-3B AWACS	JS/80842635/6953/575
196	BILLED0	2101	BILLED01	64	2	16	101	B-52H Stratofortress	JS/80842635/14ff/2
197	BILLED0	2104	BILLED04	64	4	20	102	B-52H Stratofortress	JS/80842635/14ff/6
198	FLAXON0	2105	FLAXON05	64	6	24	121	F-117 Flt-of-2	JS/80842635/6242/8009
199	FLAXON0	2105	FLAXON06	64	14	48	122	F-117 Flt-of-2	JS/80842635/6242/8010
200	SLAMMER0	2107	SLAMMER07	64	9	31	141	F15E Flt-of-2	JS/80842635/645b/7608
201	SLAMMER0	2107	SLAMMER08	64	12	39	142	F15E Flt-of-2	JS/80842635/645b/7609
202	SLAMMER1	2111	SLAMMER11	64	20	64	151	F15E Flt-of-2	JS/80842635/645b/9613
203	SLAMMER1	2111	SLAMMER12	64	17	56	152	F15E Flt-of-2	JS/80842635/645b/9614
208	SEAD/EW	2552	LEOPARD52	64	533	1421	200	EA-6B Prowler	JS/80842635/645b/34182
209	MAGPIE0	2201	MAGPIE01	64	527	1412	131	F16C Flt-of-2	JS/80842635/645b/34010
210	MAGPIE0	2201	MAGPIE02	64	524	1404	132	F16C Flt-of-2	JS/80842635/645b/34011
211	BOVINE2	5021	BOVINE21	64	515	1377	111	FA18EF Flt-of-2	JS/80842635/645b/32927
212	BOVINE2	5021	BOVINE22	64	518	1385	112	FA18EF Flt-of-2	JS/80842635/645b/32928
213	EC ISR	3431	BUZZY31	64	505	1334	401	Predator UAV	JS/80842635/645b/33752
214	EC ISR	3432	KODAK32	64	521	1398	402	Global Hawk	JS/80842635/645b/34003
218	EC Air Nodes	3461	TEA61	64	509	1353	303	E-3B AWACS	JS/80842635/645b/33728
219	EC Air Nodes	5023	SEAFARER23	64	507	1342	302	E-2C Hawkeye	JS/80842635/645b/33740
220	Unmanned Ground Sensors		UG01	61	549	1498	721	Sniper	JS/80842635/1b8c/252
221	Unmanned Ground Sensors		UG02	61	548	1491	722	Sniper	JS/80842635/1b8c/259
228	US_Convoy		US_Truck1	66	561	1518	741	Civ Medium Truck PLT	JS/80842635/1910/29
229	US_Convoy		US_Truck2	66	563	1521	742	Civ Medium Truck PLT	JS/80842635/1910/30
232	SOF_SQUAD (JTAC)		WHITE_LIGHT	61	565	1531	762	Rifle Squad	JS/80842635/1bf0/85
233	SOF_SQUAD		SOF_1L	61	566	1538	763	Rifle Squad	JS/80842635/1bf0/89
234	SOF_SQUAD		SOF_1R	61	573	1568	764	Rifle Squad	JS/80842635/1bf0/90
235	SOF_SQUAD		SOF_1G	61	572	1565	765	Rifle Squad	JS/80842635/1bf0/91
236	SOF_SQUAD		SOF_1A	61	571	1562	766	Rifle Squad	JS/80842635/1bf0/92
245	Civilian Distractors		veh_301	66	578	1590	2301	Civ Pickup	JS/80842635/6953/600
246	Civilian Distractors		veh_307	66	579	1593	2307	Civ Pickup	JS/80842635/6953/601
252	Enemy VIED		Truck_Bomb	66	555	1511	5002	Civilian Truck Bomb	JS/80842635/1910/49
253	Insurgents		Insurgent01	61	550	1505	5003	Fedayeen Team	JS/80842635/1910/113
254	Insurgents		Insurgent02	61	551	1506	5004	Fedayeen Team	JS/80842635/1910/114
258	Simulated Building		BUILDING	66	564	1525	5000	BTR-80	JS/80842635/1910/152

Figure 9: Master Entity List

Figure 9 shows a portion of the Master Entity List for this scenario. One of the challenges of integrating multiple simulations, including JSAF and GIESim, with real command and control systems such as TBMCS, is that these systems all have different ways of identifying entities and entity types. The Master Entity List serves as a "Rosetta Stone", allowing entity identifier and type information to be translated from one system to another. As shown in the figure, it includes MIDB entity names TBMCS mission numbers, callsigns, JSAF entity types and identifiers, GIESim entity identifiers, FOM entity identifiers, and HLA entity classes and identifiers.

Figure 10 shows the coverage of the various SAM systems that make up the Califon IADS, as defined in the unclassified Pacifica MIDB. The SA-10s and SA-5s are the longer-range area defense systems, while the SA-2s and SA-3s are the shorter-range point defense systems. The theater-level air superiority scenario consists of a sequence of attacks that has the objective of creating a gap in the Califon IADS in the Palmdale area, where the lower-level scenario takes place.

The Theater Battle Management Core System (TBMCS) was used to create a Target Nomination List (TNL) for this scenario. A portion of this TNL is shown in Figure 11.

The attack begins wth two B-52s each launching multiple Conventional Air Launched Cruise Missiles (CALCMs) to knock out the Margarita Peak and Santiago Peak SA-5 sites. With these long range SAMs disabled, a series of subsequent attacks against the SA-2 and SA-3 sites at Barstow, China Lake, Edwards, and Palmdale are carried out by four pairs of F-15Es and one pair of F-117s.

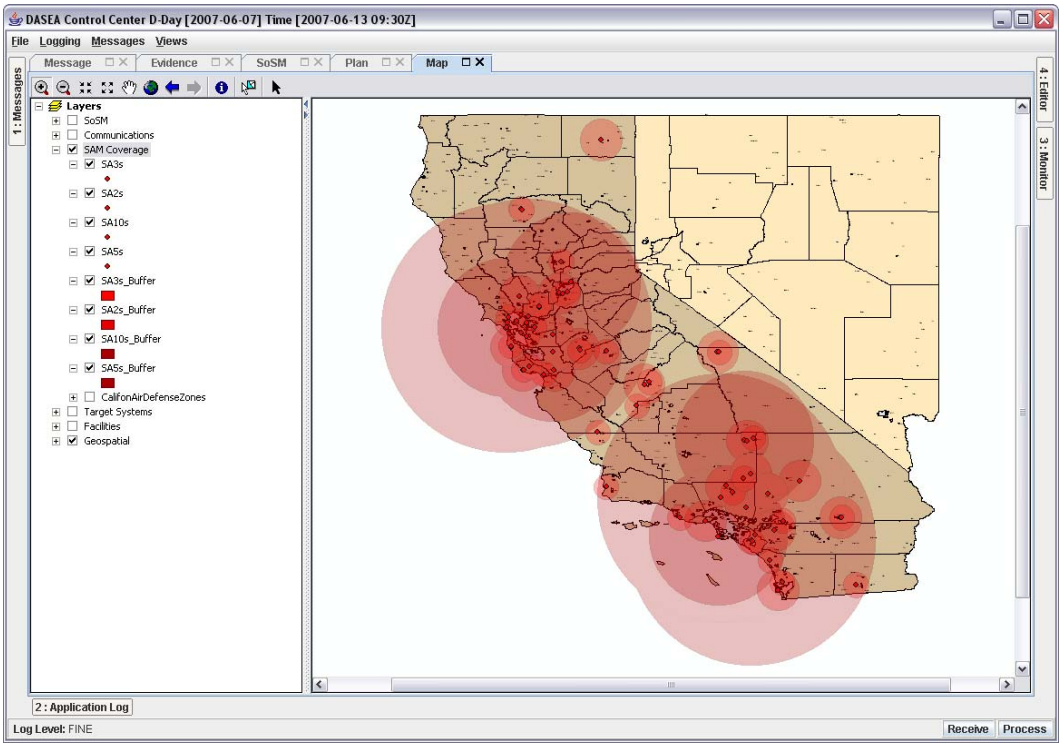

Figure 10: Califon Air Defense Coverage Areas

The other air missions in the scenario include pairs of F-16CJs and F18-Es on CAP, along with a single EA-6B Prowler. Empire Challenge 2008 aircraft include several experimental airborne networking platforms, including an E-10 prototype, Paul Revere, and the BACN prototype. Surveillance aircraft include an E3-C AWACS, an E2-C Hawkeye, an MQ-1 Predator, and an RQ-4 Global Hawk.

2.3.2 Scenario Execution

Figure 13 shows the function configuration for the execution of this scenario. JSAF generated the movements, communications and sensor emissions, and combat events, and output them using the HLA protocol. These messages are logged using JLogger in conjunction with JSAF. These messages are passed to the Viewer via the HLA-to-XML gateway, where they are displayed dynamically. The HLA messages also pass through the DIS-HLA Gateway, where they are translated from the HLA protocol to the DIS protocol. This allows them to be read by the SIMPLE application, which translates the aircraft movements into a Link-16 message track format. These are fed through MTDS to TBMCS, where they are used to update the track database, and are displayed on the COP.

1. MARGARITA PEAK SA-5 SITE 0992MB0002 - DD001

Desired Effects:
Rationale:
Collateral Concerns:

1	SA5 LNCHR	Unknown	332638.00N 1172256.00W	Unknown	Unknown	
2	SA5 LNCHR	Unknown	332638.00N 1172310.00W	Unknown	Unknown	
3	SA5 LNCHR	Unknown	332638.00N 1172322.00W	Unknown	Unknown	
4	SA5 LNCHR	Unknown	332638.00N 1172336.00W	Unknown	Unknown	
5	SA5 LNCHR	Unknown	332638.00N 1172348.00W	Unknown	Unknown	
6	SA5 LNCHR	Unknown	332638.00N 1172402.00W	Unknown	Unknown	
7	SA5 LNCHR	Unknown	332634.00N 1172300.00W	Unknown	Unknown	
8	SA5 LNCHR	Unknown	332634.00N 1172310.00W	Unknown	Unknown	
9	SA5 LNCHR	Unknown	332634.00N 1172326.00W	Unknown	Unknown	
10	SA5 LNCHR	Unknown	332634.00N 1172332.00W	Unknown	Unknown	
11	SA5 LNCHR	Unknown	332634.00N 1172352.00W	Unknown	Unknown	
12	SA5 LNCHR	Unknown	332634.00N 1172358.00W	Unknown	Unknown	
13	TALL KING RDR	Unknown	332616.56N 1172328.56W	Unknown	Unknown	
14	PERFECT PATCH VAN	Unknown	332614.56N 1172332.56W	Unknown	Unknown	
15	SQUARE PAIR RDR	Unknown	332614.56N 1172336.56W	Unknown	Unknown	
16	SQUARE PAIR RDR	Unknown	332612.56N 1172324.56W	Unknown	Unknown	
17	SPT BLDG	Unknown	332612.00N 1172328.00W	Unknown	Unknown	
18	GUIDANCE TRACKING BLDG	Unknown	332612.00N 1172330.00W	Unknown	Unknown	Unknown
19	GENERATOR	Unknown	332612.00N 1172332.00W	Unknown	Unknown	
20	BACK NET RDR	Unknown	332610.00N 1172334.00W	Unknown	Unknown	
21	ODD PAIR RDR	Unknown	332610.00N 1172336.00W	Unknown	Unknown	

Figure 11: Target Nomination List

Title: SIMULATION EXERCISE 08-1 (SE 08-1) Scenario Overview

	MSN Type	MSN	Support	Unit	NOTYPAC	C/S	ICAO	SCL	Air Location	Vul/Tgt Time(s)	TGT	ALT	AR Track	ARCT	Off load	Control Agency	TACP
Internal	OCA	2104		22BS	1XB52H	BILLED04	KIND	**15A86**	LAR CAP-N	1400Z/1405Z	Margarita Peak SA-5					72 ACF CRC 4	
	OCA	2101		22BS	1XB52H	BILLED01	KIND	**15A86**	LAR CAP-L	1400Z/1405Z	Santiago Peak SA-5					CRC-4	
	OCA	2105		35FS	2XF117	FLAXON05	KDPG	**2XG10**		1435Z/1440Z	Barstow SA-2					CHALICE	
	OCA	2107	SEAD/EW	362FS	2XF15E	SLAMMER07	KLSV	**2G31X2G10**		1415Z1420Z	China Lake Inyokern SE Sa-2					CHALICE	
	OCA	2111	SEAD/EW	362FS	2XF15E	SLAMMER11	KLSV	**2G31X2G10**		1440Z1445Z	Edwards NE SA-2					CHALICE	
	OCA	2113	SEAD/EW	362FS	2XF15E	SLAMMER13	KLSV	**2G31X2G10**		1415Z1420Z	China Lake NWC SA-3					CHALICE	
	OCA	2115	SEAD/EW	362FS	2XF15E	SLAMMER15	KLSV	**2G31X2G10**		1440Z1445Z	Palmdale SA-3					CHALICE	
	SEAD	2201		77FS	2XF16CJ	MAGPIE01	KLSC	**22A88X2W2**	CAP-J	1410Z/1515Z						CHALICE	
	EW	2552		VMAQ2	1XEA6B	LEOPARD52	KDMA	**E3Q99X1A88X1**	CAP-E	1410Z/1515Z						CHALICE	
EC08	XCAS	5021		VMF333	2XF18E	BOVINE21	KDPG	**FG31X2IDX2W2**	CP SE08	1420Z-1530Z		280				CHALICE	White Lightning
	OTR	3411	E-10	767RS	1XB767	PYTHON11	KDMA	**BEST**	EC08 YELLOW	1420Z-1530Z		300				CHALICE	
	OTR	3412	Paul Revere	768RS	1XB767	FISHLIPS12	KDMA	**BEST**	EC08 GREEN	1420Z-1530Z		320				CHALICE	
	OTR	3403	BACN	769RS	1XC9	ELVIS03	KDMA	**BEST**	EC08 BLUE	1420Z-1530Z		200				CHALICE	
	XAEW	3461	AWACS	964ACS	1XE3C	TEA61	KLSC	**BEST**	EC08 YELLOW	1420Z-1530Z		320				CHALICE	
	XAEW	5023	Hawkeye	VAW126	1XE2C	SEAFARER23	CV70	**BEST**	EC08 GREEN	1420Z-1530Z		240				CHALICE	
	RECCE	3431	Pred	770RS	1XMQ1	BUZZY31	KLSV	**2XAGM114**	EC08 GREEN	1420Z-1530Z		200				CHALICE	
	RECCE	3432	Global Hawk	771RS	1XRQ4	KODAK32	KLSV	**BEST**	EC08 YELLOW	1420Z-1530Z		200				CHALICE	

Figure 12: Mission List

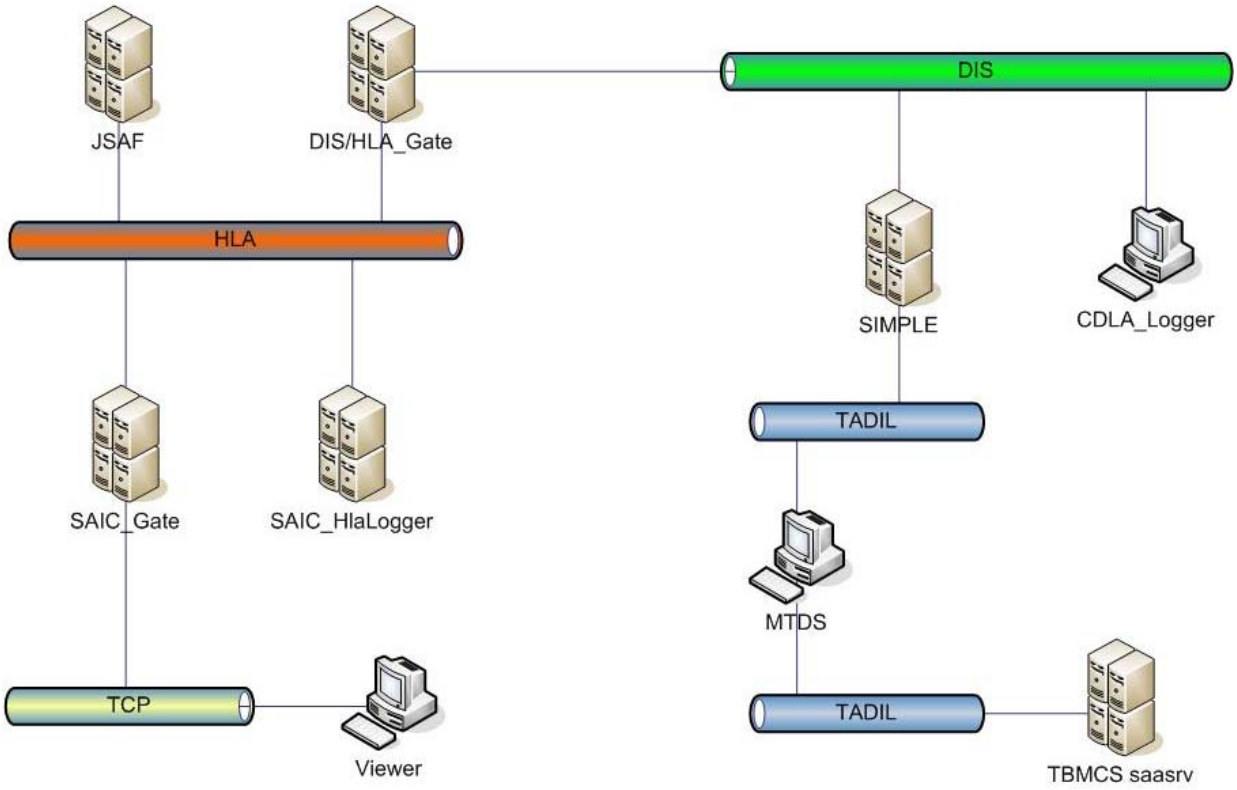

Figure 13: Functional Configuration

Figures 14 through 18 illustrate selected aspects of the scenario execution. Figure 14 shows the entire Califon IADS modeled as JSAF entities. Figure 15 shows the locations of all entities at the start of the scenario. The two horizontal lines of symbols in the lower right part of the screen represent the two B-52s and the sequence of CALCMs that each launches. The Califon IADS elements are shown in red in the lower-left part of the screen. Blue strike aircraft, ISR aircraft, airborne networking aircraft, and UAVs are shown in blue in the upper left and center areas of the screen. Figure 16 shows a more detailed view of the two B-52s launching their cruise missiles. The two horizontal lines of black symbols represent the planned release points for each of the eight missiles launched by each B-52. The B-52s themselves are shown in blue. The cruise missiles in flight are shown as short horizontal lines spread out to the west of the launch points.

Figure 17 shows a more detailed view of the Santiago Peak SA-5 site, showing the distribution of the TELs, radars, and command and control vehicles at the site. Elements of other SAM sites can be seen to the northeast and northwest.

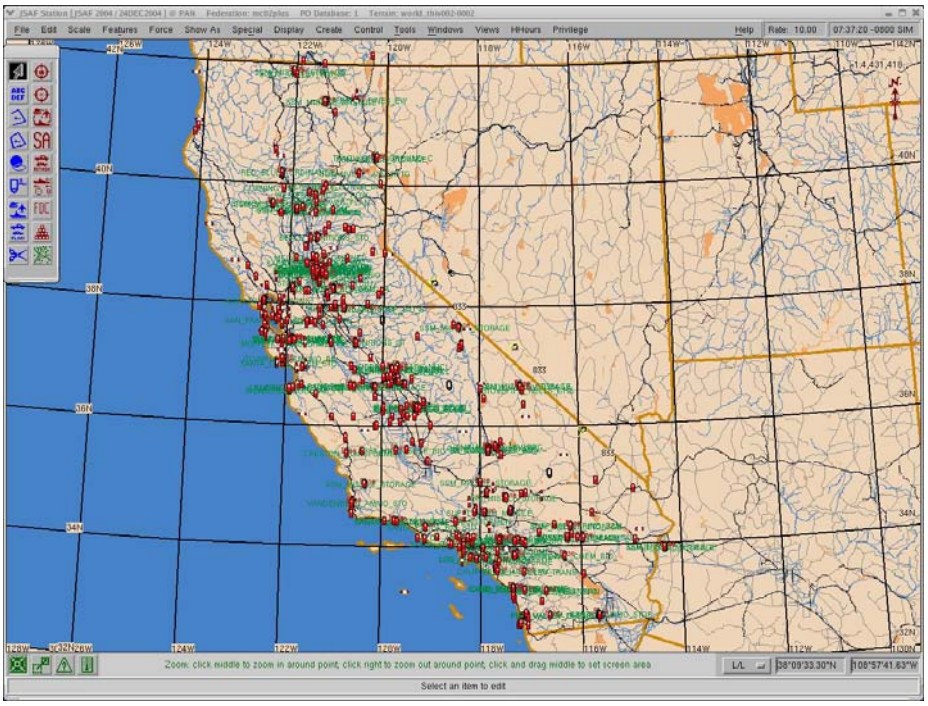

Figure 14: Califon IADS Modeled in JSAF

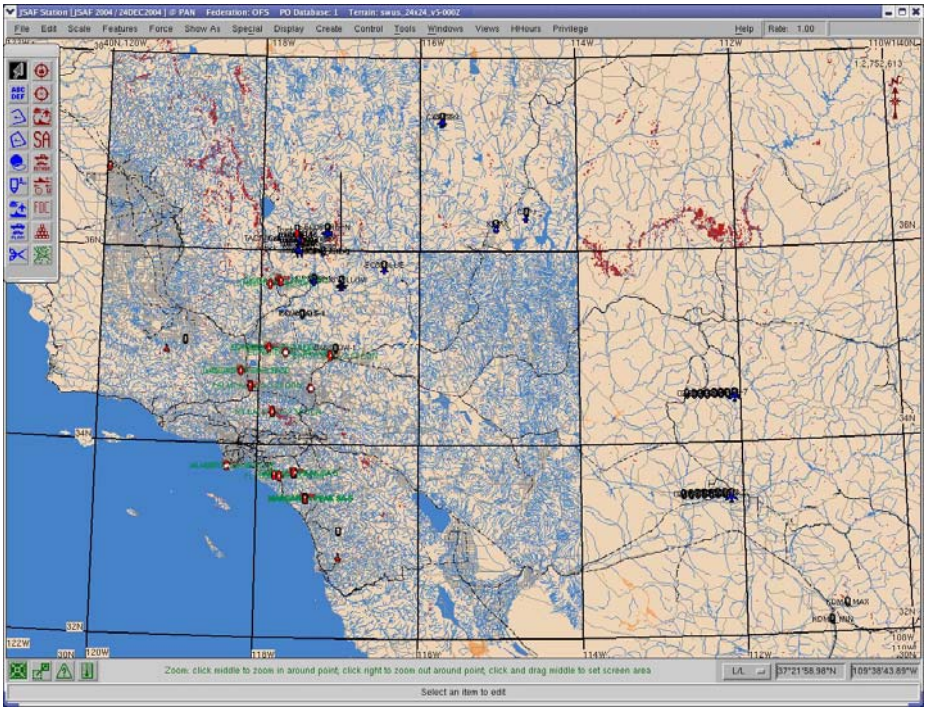

Figure 15: Initial Entity Locations

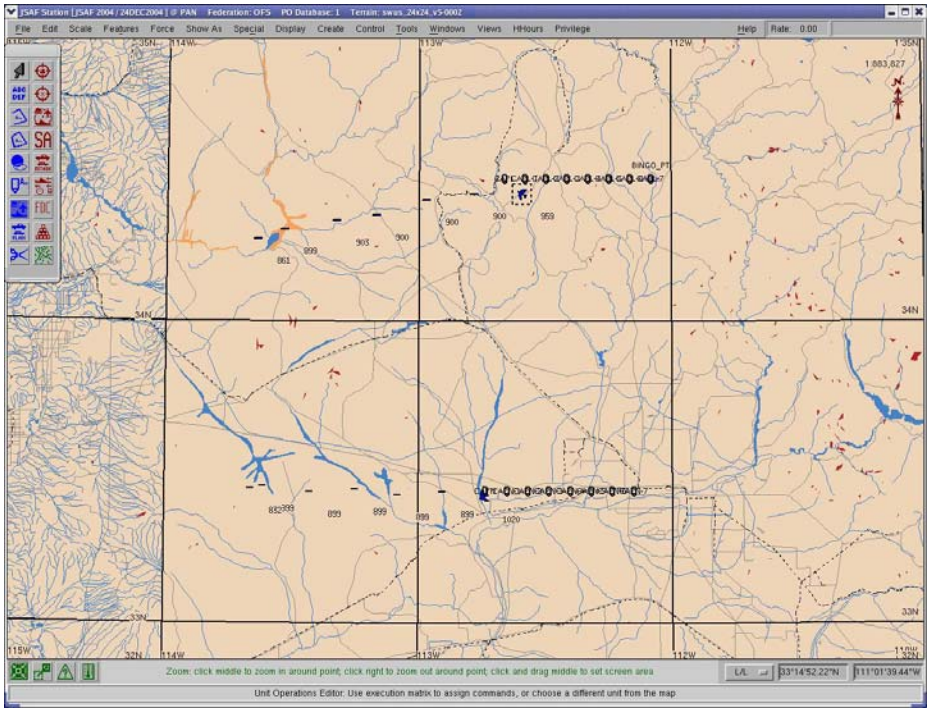

Figure 16: Cruise Missiles in Flight

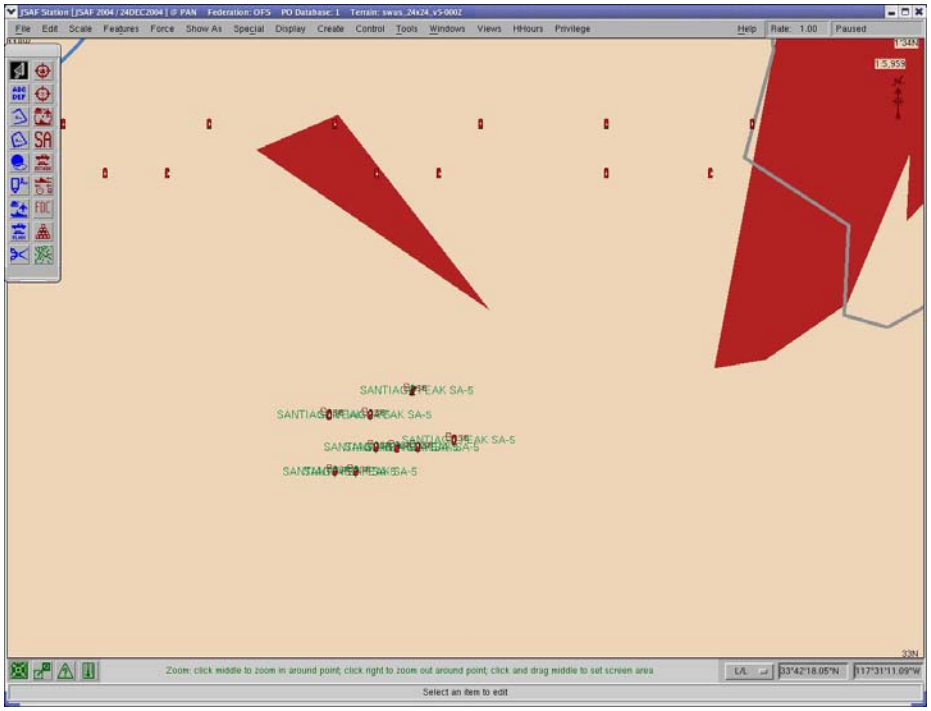

Figure 17: Santiago Peak SA-5 Site

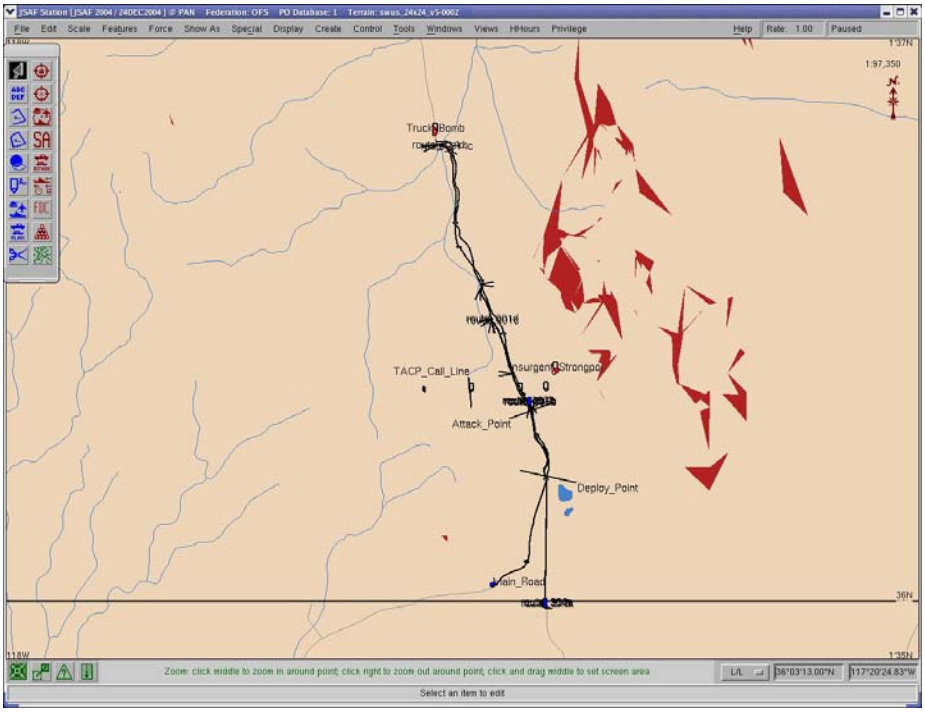

Figure 18: Ground Scenario Locations

Figure 18 shows the area where the ground scenario takes place. A US truck convoy moves up the route shown from south to north. Civilian traffic moves randomly along these roads as well. The insurgent truck bomb vehicle moves down the road from north to south. As it moves, the truck bomb is detected moving toward the convoy by a series of UGS spaced along the road. These reports are communicated to the Joint AOC, which tasks the nearby Predator to monitor the suspicious-looking truck. The truck stops as it nears the approaching convoy and the insurgents dismount and hide nearby. The Predator operator sees the stopped truck. However, before the convoy can be warned, the insurgents detonate the truck bomb as the convoy passes, destroying one of the convoy trucks. The insurgents retreat toward a nearby building. However, before they can reach the building, they are spotted by a nearby SOF squad. The JTAC who is attached to the SAF squad calls in an air strike against the building. Two nearby F-18Es are dispatched to carry out the attack. They coordinate their attack with the JTAC to minimize the possibility of civilian casualities. A few minutes later, the F-18Es strike the building, killing the insurgents. The Predator provides follow-up BDA.

2.3.3 Scenario Analysis

The logged HLA entity state and interaction messages were analyzed to identify and extract the key scenario events that resulting in the failure to identify the threat and warn the convoy before the truck bomb could be detonated.

The HLA entity state and interaction messages were exported from JLogger as comma-separated-value (CSV) files by message type, and were then imported into a Microsoft Access database. They were then processed to make the linkages between the various message types clear. The sequence of processing steps performed on the logged HLA message traffic was as follows:

1) Extracted lists of distinct entities of each class – aircraft, ground vehicles, humans, munitions, etc.
2) Created a master entity list to allow identifiers and type information to be translated among:
   - TBMCS mission ids & callsigns
   - MIDB BE numbers, OSuffixes, and DMPI identifiers
   - HLA entity identifiers and classes
   - JSAF entity identifiers, types, and markings
   - GIESim entity identifiers
3) Added callsigns to all logged HLA entity state and interaction records for readability
   - Aircraft, Ground Vehicles, Humans, etc.
   - Blip (detection/track) entities – sensing and detected entities
   - Weapon Fire, Munition Detonation, & Damage Assessment interactions – firing, target, and munition entities
   - GIESim Send, Receive, and Entity State interactions – sending and receiving entities

Key events were then identified and extracted, as follows:

- Movement
  - Changes to PhysicalEntity class WorldLocation, and VelocityVector attribute values; reflecting movement starts and stops
- Communication
  - GIESIM_MSG_SEND interactions, including links to the sending and receiving entities
  - GIESIM_MSG_RCVD interactions, including links to the receiving entity
  - RadioTransmitter class, identifying radio transmitters and their attributes

- Sensors
  - Blip class, describing radar and sensor detections
  - EmitterSystem class, identifying radars and active sensors and their attributes
  - JammerBeam class, identifying jammer emissions and their attributes
  - RadarBeam class, identifying individual radar emissions and their attributes
- Combat
  - WeaponFire interactions, describing weapon firing events, including links to the firing entity, optionally to the target entity, and optionally to the munition entity
  - MunitionDetonation interactions, describing weapon detonation events, including links to the firing entity, optionally to the target entity, and optionally to the munition entity
  - DamageAssessment interactions, evaluating the damage inflicted on an entity by a weapon detonation event, including links to the firing entity, optionally to the target entity, and optionally to the munition entity
  - Changes to PhysicalEntity class DamageState attribute values, indicating that an entity was damaged or destroyed

A summary of the key scenario events is shown in Figure 19. The individual events are color-coded by type, as shown at the top of the table. The event types include:

- Movement events – showing when the truck bomb vehicle started moving, stopped moving, and was destroyed by its detonation
- Detection events – showing when the various UGS and civilian vehicles first saw the truck bomb vehicle, and when they last saw the truck bomb vehicle
- Damage assessment events – showing when the truck bomb and the convoy truck were destroyed, as well as how another truck in the convoy escaped damage, even though it was within the potential damage radius of the bomb

Timestamp	Event Type	Moving Entity	Damage State	PowerPlantOn	
Timestamp	Event Type	Seeing Entity	Sensor	Seen Entity	
Timestamp	Event Type	Seeing Entity	Sensor	Seen Entity	
Timestamp	Event Type	Seeing Entity	Sensor	Seen Entity	
Timestamp	Event Type	Firing Entity	Munition Type	Target Entity	Computed Result
1212168972.561214	Initial	Truck_Bomb	NoDamage	Yes	
1212172317.569577	Start Moving	Truck_Bomb	NoDamage	Yes	
1212172458.541548	First Seen	UG02	SPOT_SAR	Truck_Bomb	
1212172550.443013	First Seen	UG03	SPOT_SAR	Truck_Bomb	
1212172817.025066	First Seen	UG04	SPOT_SAR	Truck_Bomb	
1212172995.955829	First Seen	veh_301	SPOT_SAR	Truck_Bomb	
1212173001.473677	First Seen	veh_310	SPOT_SAR	Truck_Bomb	
1212173002.565603	First Seen	UG06	SPOT_SAR	Truck_Bomb	
1212173016.594798	First Seen	veh_317	SPOT_SAR	Truck_Bomb	
1212173033.893772	First Seen	veh_307	SPOT_SAR	Truck_Bomb	
1212173035.817063	First Seen	veh_308	SPOT_SAR	Truck_Bomb	
1212173071.243725	Stop Moving	Truck_Bomb	NoDamage	No	
1212173090.686806	Last Seen	UG02	SPOT_SAR	Truck_Bomb	
1212173118.535735	Last Seen	UG03	SPOT_SAR	Truck_Bomb	
1212173275.069619	Last Seen	UG04	SPOT_SAR	Truck_Bomb	
1212173280.530691	Last Seen	veh_310	SPOT_SAR	Truck_Bomb	
1212173301.252892	Last Seen	veh_307	SPOT_SAR	Truck_Bomb	
1212173305.830832	Last Seen	veh_317	SPOT_SAR	Truck_Bomb	
1212173332.312785	Last Seen	veh_301	SPOT_SAR	Truck_Bomb	
1212173348.547111	Last Seen	veh_308	SPOT_SAR	Truck_Bomb	
1212174020.123048	Damage Assessment		munition_US_Mk82	Truck_Bomb	Catastrophic
1212174020.123693	Damage Assessment			US_Truck4	NoDamage
1212174020.132715	Damage Assessment			US_Truck1	Catastrophic
1212174020.446691	Destroyed	Truck_Bomb	Destroyed	Yes	
1212174295.101663	Last Seen	UG06	SPOT_SAR	Truck_Bomb	

Figure 19: Key Scenario Events

3 JVIEW COORDINATE, ORIENTATION, AND VECTOR CONVERSION SERVICES

This task was focused on the development of an enhanced coordinate conversion capability for AFRL's JView 3D visualization package, based on the integration of National Geospatial-Intelligence Agency's (NGA's) GEOTRANS coordinate conversion software and the Synthetic Environment Data Representation and Interchange Specification (SEDRIS) Spatial Reference Model (SRM) software. This capability provides conversion between geodetic, geocentric, local Cartesian, Mercator, Transverse Mercator, Polar Stereographic, Lambert Conformal Conic, Universal Transverse Mercator (UTM), Universal Polar Stereographic (UPS), and Military Grid Reference System (MGRS) coordinates. It also provides datum transformation between WGS84 and all of the other local and global datums currently supported by NGA. It provides conversion between WGS84 ellipsoid heights and EGM96 geoid (gravity-based) heights. In addition to converting position coordinates, it incorporates the orientation and vector conversion capabilities recently added to the SEDRIS SRM to support the Test and Training Enabling Architecture (TENA) program. This allows the software to be used by various simulation applications to convert entity state information, including position, velocity, and acceleration, between different spatial reference frames. Additional requirements include high-performance (~10K to 100K points per second), sub-meter accuracy, and thread safety.

3.1 Background

The GEOTRANS software is the standard coordinate conversion and datum transformation software used throughout DoD, as well as by a worldwide user community. GEOTRANS was originally developed by the US Army Topographic Engineering Center (TEC) in 1997, using the C programming language, and in accordance with the software reuse guidelines published by the Army Reuse Center. In 1999, it was adopted by the NGA, and expanded to meet the requirements of the Joint Mapping Tool Kit (JMTK) program. In 2001, it was first made publicly available by NGA. The GEOTRANS software continues to be included within NGA's Commercial Joint Mapping Tool Kit (C/JMTK).

In 2006, maintenance of the GEOTRANS software was made part of NGA's Mensuration Services Program (MSP), where it was adapted to serve as the MSP Coordinate Conversion Service. Northrop Grumman Information Systems continued to maintain the GEOTRANS software under this NGA program. The current publicly available version is GEOTRANS v2.4.2. It is available from NGA's web site at: http://earth-info.nga.mil/GandG/geotrans/.

The GEOTRANS software consists of an interactive application, supported by an underlying software library. Figure 20 shows the GEOTRANS application GUI, which allows coordinates to be converted between any two supported combinations of coordinate reference systems and datums. In conjunction with geodetic coordinates, height values can be converted between WGS84 ellipsoid height, and several variations of geoid height, based on the EGM96 and EGM84 gravity models.

Figure 21 shows the underlying structure of the GEOTRANS software. The GEOTRANS application is supported by the GEOTRANS engine, which provides an API that can be used to convert points from any combination of supported datum and coordinate system to any other supported combination. It also supports the creation (and deletion) of user-defined ellipsoids and datums. The GEOTRANS Engine is supported by a collection of thirty-six individual C modules, one for each type of coordinate system or map projection supported, plus modules that provide access to ellipsoid, datum, and geoid model data. While these individual modules are largely independent of one another, there are a few dependencies. For example, the MGRS module depends on the UTM and UPS modules, which in turn depend on the Transverse Mercator and Polar Stereographic modules.

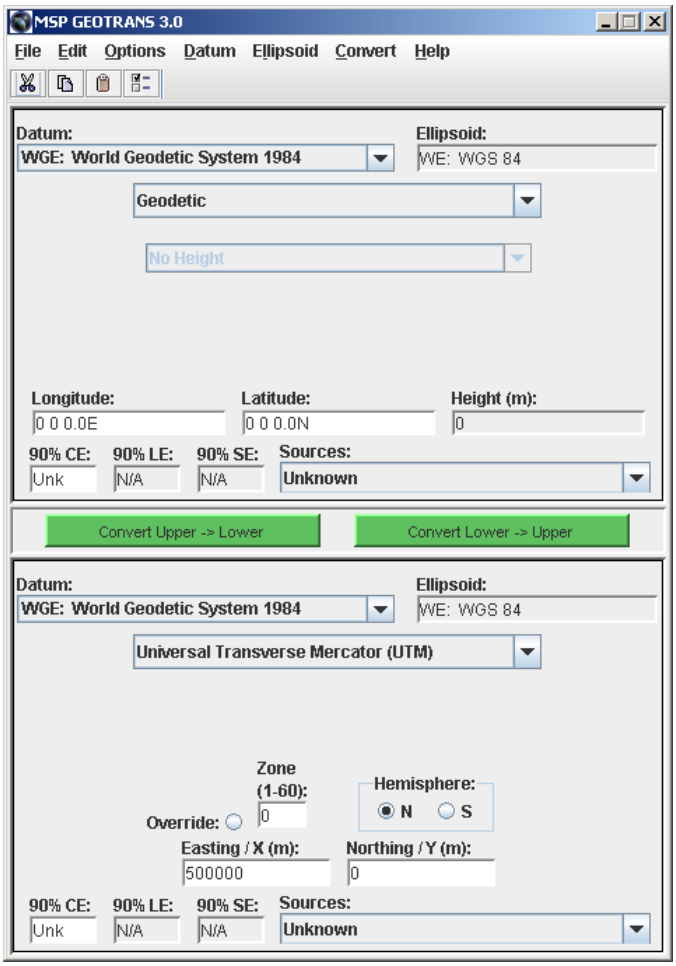

Figure 20: GEOTRANS Application GUI

Under the MSP program, the original C implementation was replaced by an object-oriented implementation in C++. The GEOTRANS Engine and the individual C modules have been converted into corresponding C++ classes. The API of the Coordinate Conversion Service class is now thread safe. The classes that support individual coordinate system or map projection types are all derived from a single abstract parent class. Supporting class hierarchies for coordinate tuples, and for coordinate system parameter sets, have also been defined.

A number of users and programs have requested a Java implementation of GEOTRANS. However, NGA has not yet funded such an implementation. In late 2007, a partial Java implementation of GEOTRANS was created to support the Joint Targeting Toolbox (JTT) program. This is known as the Targeting Database Access Layer (TDAL) Coordinate Conversion capability. This implementation included a specialized version of the GEOTRANS engine, which takes as input a point specified using any of the coordinate formats used in the MIDB, and converts that point to all of the other formats. This implementation explicitly supports the conversion of points between geodetic, geocentric, UTM, UPS, and MGRS coordinates. It also implicitly supports Transverse Mercator and Polar Stereographic coordinates, since they underlie UTM and UPS.

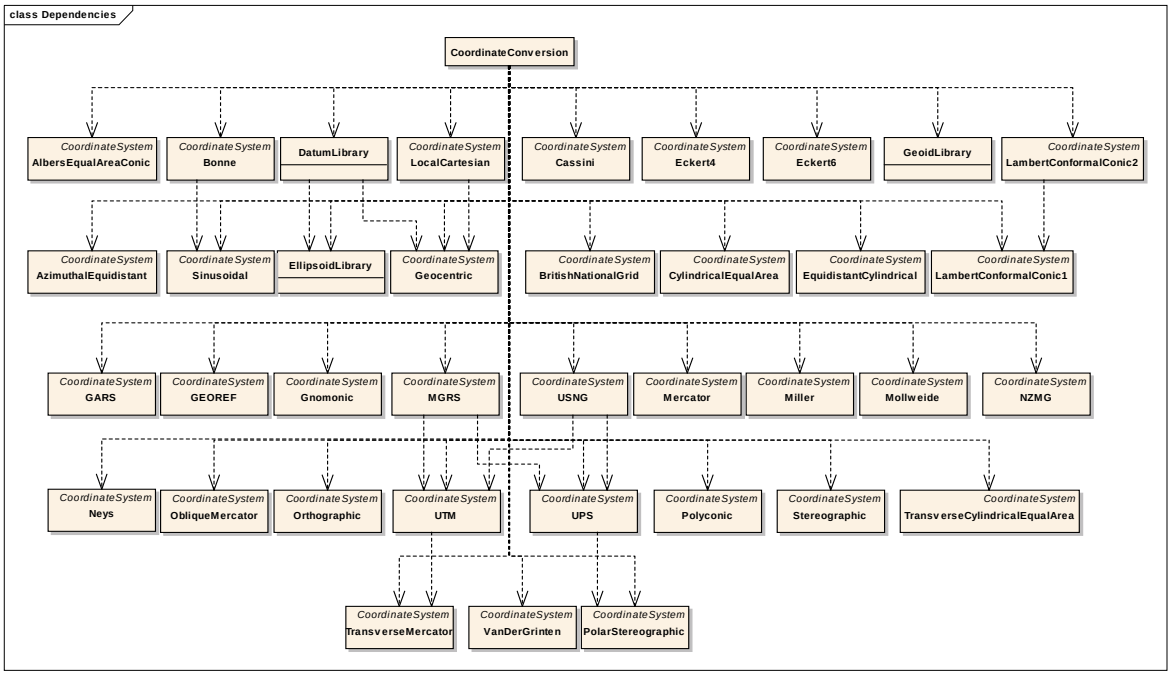

Figure 21: GEOTRANS Software Architecture

The SEDRIS Spatial Reference Model (ISO 18026) also specifies a coordinate conversion capability. Recently, sponsored by the TENA program, it has been augmented to also support the conversion of orientation and vector information, allowing entity state information, including position, orientation, velocity, and acceleration, to be converted between different spatial reference frames.

This task leveraged the GEOTRANS v3.0 implementation, the JTT TDAL coordinate conversion implementation, and the SEDRIS SRM v4.4 implementation to create a thread-safe, high-performance, Java coordinate, orientation, and vector conversion software utility for AFRL's JView 3D visualization toolkit. The requirements for the JView Coordinate Conversion Service software were as follows:

1) Provide coordinate conversion between any of the coordinate system types listed in Table 2.

Table 2: Required Coordinate System Types

Coordinate System	Type	Reference
Geodetic	Three Dimensional	NGA TR 8350.2
Geocentric	Three Dimensional	NGA TR 8350.2
Local Cartesian	Three Dimensional	Note 1
Lambert Conformal Conic (with one or two standard parallels)	Map Projection	SNYDER
Mercator	Map Projection	SYNDER
Polar Stereographic	Map Projection	SNYDER
Transverse Mercator	Map Projection	SNYDER
Universal Polar Stereographic (UPS)	Grid	NGA TM 8358.2
Universal Transverse Mercator (UTM)	Grid	NGA TM 8358.2
Military Grid Reference System (MGRS)	Grid	NGA TM 8358.1

Note: Local Cartesian is a 3D Cartesian coordinate system with its origin at a point specified in geodetic coordinates. Normally, the XY plane is tangent to the ellipsoid surface at the origin, while the Z axis is normal to the ellipsoid surface at the origin.

2) Provide conversion of orientations between any of the three-dimensional coordinate system types listed in Table 2, supporting the following orientation representations:
   a. 3x3 rotation matrix,
   b. Axis-angle,
   c. Euler angles ZXZ,
   d. Tait-Bryan angles, i.e., roll, pitch, and yaw, and
   e. Quaternions.
3) Provide conversion of 3D vectors between any of the three-dimensional coordinate system types listed in Table 2.
4) Provide datum transformations between any of the datums listed in NGA TR8350.2.

5) Provide height conversions between:

   a. Geodetic Height relative to the surface of the WGS84 ellipsoid,

   b. Orthometric Height relative to the Earth Gravity Model 1996 (EGM96) with 15 minute by 15 minute grid spacing and bilinear interpolation.

6) Provide integrated coordinate conversions of points, orientations, and vectors, datum transformations, and height conversions in any combination as specified above.

7) Target performance shall be a minimum of 10,000 points per second, with a goal of 100,000 points per second.

8) Target accuracy shall be a minimum of 1 meter, with a goal of 1cm. (Note that this refers to computational accuracy only. Real world location accuracy depends on the accuracy of the input data, as well as a number of other factors.)

9) The implementation shall be in Java, with a thread safe API. The target platform is therefore irrelevant, except in that it impacts the achievement of the performance goals stated above.

3.2 Implementation

In order to implement the requirements listed above, two software components were developed:

1) The JView Coordinate Conversion Service (JView CCS) – this component provides coordinate conversion, datum transformation, and height conversion between any of the coordinate reference frame types listed in Table 2.

2) The JView Orientation and Vector Conversion Service (JView OVCS) – this component provides orientation and vector conversion between any of the coordinate reference frame types listed in Table 2. It depends on the JView CCS component to define the source and target coordinate reference frames.

To support the interactive use and testing of these two software components, corresponding applications were also developed.

3.2.1 JView Coordinate Conversion Service

The design of the JView Coordinate Conversion Service parallels that of NGA's MSP Coordinate Conversion Service. The primary difference is that a smaller set of coordinate system types are supported.

The public API of the JView Coordinate Conversion Service is shown in Figure 22. It consists of the CoordinateConversionService class, and its subordinate DatumLibrary and EllipsoidLibrary classes, as well as hierarchical sets of CoordinateSystemParameter and CoordinateTuple classes that are used as parameters. The class CoordinateConversionService provides the majority of the functionality. Its constructor sets the initial state of the coordinate conversion service in preparation for coordinate conversion and/or datum transformation operations, specifying the source datum, source coordinate system type and parameters, if any, target datum, and target coordinate system type and parameters, if any. Once a CCS object has been created, it can be used to perform the specified conversion, or its inverse, on any number of coordinate tuples, either individually or collectively. However, for thread safety reasons, it cannot be used to perform a different conversion operation. A separate CCS object instance must be created for each distinct conversion operation to be performed.

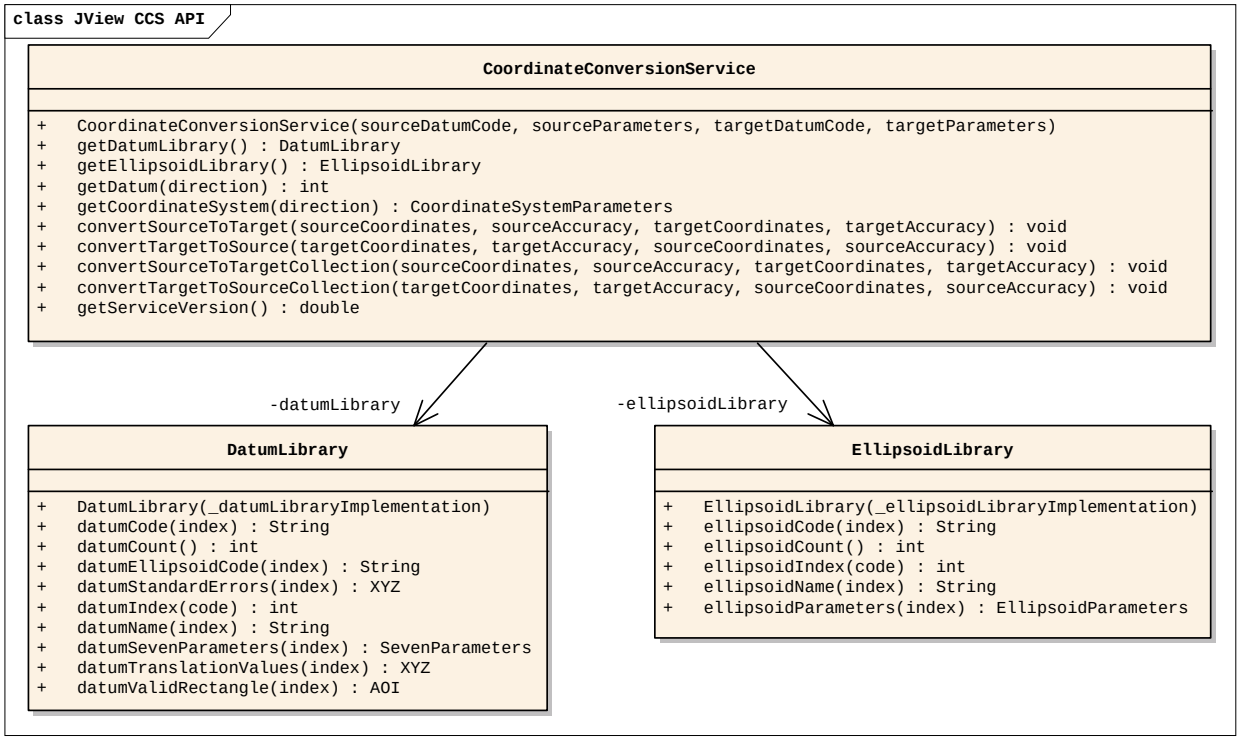

Figure 22: JView CCS Public API

The method convertSourceToTarget converts a single specified source coordinate tuple, referenced to the source coordinate system and datum, into an equivalent target coordinate tuple, referenced to the target coordinate system and datum. Accuracy information for the source coordinate tuple can be optionally specified, and, if present, is used to estimate the accuracy of the output coordinate tuple. The method converTargetToSource performs the inverse operation, converting from the target coordinate system and datum to the source coordinate system and datum.

The methods convertSourceToTargetCollection and convertTargetToSourceCollection are similar, but operate on generic Java collections of coordinate tuples. This allows large numbers of coordinates to be converted with a single method call.

The methods getDatum and getCoordinateSystem allow the source or target datum, or the source or target coordinate system type and parameters, respectively, of the CCS object to be retrieved.

The methods getDatumLibrary and getEllipsoidLibrary allow those subordinate objects to be accessed. They provide methods that allow more detailed information on datum and ellipsoid parameters to be accessed.

Finally, the method getServiceVersion returns the version number of the CCSobject.

The JView Coordinate Conversion application is called JView GEOTRANS 3.0J, and is a customized version of the NGA MSP GEOTRANS 3.0 application. Its GUI is shown in Figure 23. Except for the more limited lists of coordinate systems and vertical datums that it provides, it is identical to the GEOTRANS 3.0 application GUI. The implementation is somewhat different internally, however, as it accesses the JView CCS implementation, which is written in Java, while the NGA MSP GEOTRANS 3.0 application uses a Java Native Interface (JNI) layer to access the MSP CCS implementation, which is written in C++.

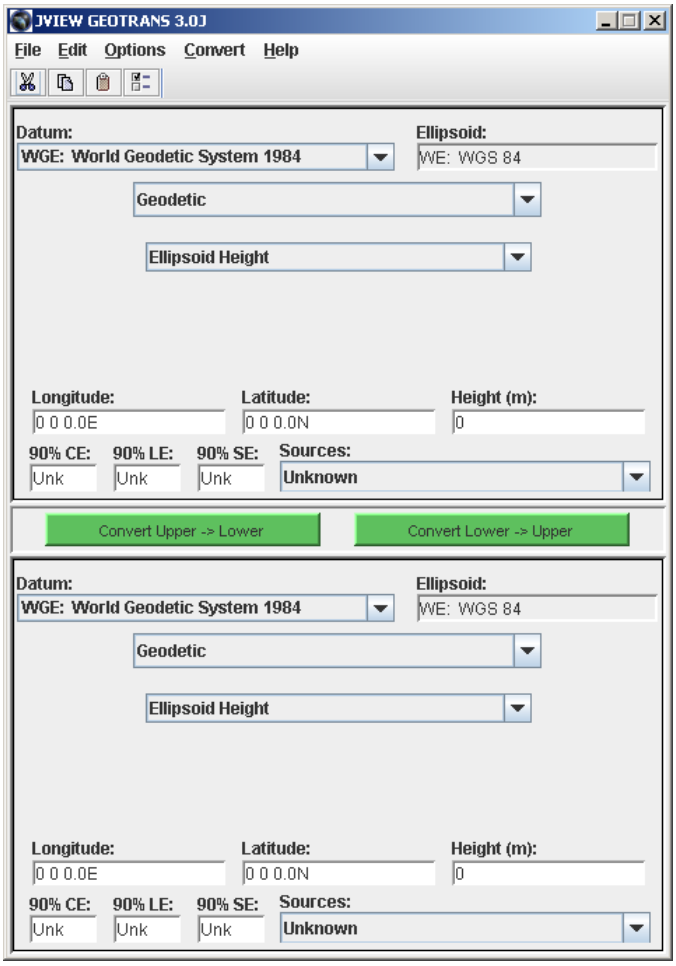

Figure 23: JView GEOTRANS 3.0J Application GUI

Accuracy testing of the JView Coordinate Conversion Service software was performed using a subset of the test cases and test procedures developed for NGA's MSP GEOTRANS v3.0 application and Coordinate Conversion Service. These test cases and test procedures were used in the successful Final Acceptance Testing of the MSP GEOTRANS v3.0 application and Coordinate Conversion Service in July 2009. The tests cases are organized as follows:

1) Test procedures for testing the JView GEOTRANS 3.0J application, using the application GUI,

2) Test procedures using a collection of spreadsheets containing more than 200,000 test cases supplied by NGA for testing GEOTRANS in 1999; which address all supported coordinate system types and datums; these test cases are executed using a test application ("the spreadsheet tester") which reads the spreadsheets, performs the test cases, and writes the results back to the spreadsheets for comparison with the expected results,

3) Test procedures using a collection of test cases developed by the NGA Coordinate System Analysis Team (CSAT), known as the "NGA gold data set", which address a core subset of coordinate system types that closely corresponds to those listed in Table 2; these test cases are executed using the file processing functionality of the JView GEOTRANS 3.0J application, and

4) Test procedures for thread safety; these test cases are executed using a test application ("the thread-safety tester") which concurrently performs the same set of coordinate conversion operations using multiple threads, and compares the results produced by each thread with a set of expected results.

In August 2009, the subset applicable to the coordinate system types listed in Table 2 was used to test the JView Coordinate Conversion Service software. The results of these tests were identical to those of the MSP FAT testing, showing that the accuracy requirement of no more than 1 meter of computation error was successfully achieved.

Performance testing of the JView Coordinate Conversion Service software was performed using a test set containing 49,472 geodetic coordinate points distributed world-wide. This collection of points consists of two subsets:

1) A 10-degree world-wide latitude-longitude grid, and

2) A collection of national political borders and coastlines derived from NGA's 1:1,000,000 scale Digital Chart of the World (DCW) dataset.

This data set was converted to each of the coordinate system types listed in Table 2, using the file processing functionality of the JView GEOTRANS 3.0J application. Timing was performed using the Java Joda-Time utility (available at http://joda-time.sourceforge.net/). The elapsed time required to process the entire dataset was converted into a points per second (PPS) value. These are shown in Table 3.

For those coordinate system types, particularly map projections, that are not world-wide in extent, errors were generated for those points located outside the valid extent of the coordinate system. For example, for UTM, all of the test points located in the Polar Regions caused errors. Conversely, for UPS, all of the points located outside the Polar Regions caused errors. The error handling associated with these points had a significant negative impact on performance for these coordinate system types. However, this could easily be addressed by tailoring the dataset for each coordinate system type to include only valid points.

Table 3: Coordinate Conversion Performance

Coordinate System	Points Per Second	Number of Errors
Geodetic	197,888.00	0
Geocentric	166,572.39	0
Local Cartesian	150,829.27	0
Lambert Conformal Conic (with one or two standard parallels)	158,564.10	398
Mercator	158,564.10	796
Polar Stereographic	90,442.41	14,656
Transverse Mercator	68,902.51	20,276[1]
Universal Polar Stereographic (UPS)	43,975.11	47,707
Universal Transverse Mercator (UTM)	117,232.23	1314
Military Grid Reference System (MGRS)	67,400.54	0

In spite of the impact of error handling, the performance of the JView Coordinate Conversion Service software was very impressive, exceeding the goal of 100,000 points per second in most cases. It was faster than the MSP GEOTRANS 3.0 C++ implementation for all coordinate system types, in some cases up to five times faster. This appeared to be due to the adaptive optimization performed within the Java Virtual Machine, which dynamically analyzes the performance of the code during its execution, and selectively recompiles portions of the code to improve its performance.

3.2.2 JView Orientation and Vector Conversion Service

The design of the JView Orientation and Vector Conversion Service is loosely based on the design of the SEDRIS SRM v4.4 orientation and vector conversion functionality. However, the detailed design has been modified for greater compatibility with the GEOTRANS v3.0 API.

The JView OVCS API is shown in Figure 24. It primarily consists of the OrientationAndVectorConversionService class, the Orientation and Vector3D classes, and the five orientation parameter classes.

The OrientationAndVectorConversionService (OVCS) class provides public methods that can be used to convert orientations and vectors between two 3D coordinate reference frames. The constructor takes a CoordinateConversionService (CCS) object as a parameter. This CCS object must be constructed with the appropriate source and target 3D coordinate reference frames. The OVCS object uses this CCS object to obtain information about the source and target coordinate reference frames, and to perform any required coordinate conversions between them.

[1] For Transverse Mercator, there were also 26,635 points that produced warnings, due to the distance of the point from the central meridian of the projection.

The methods convertOrientation and convertOrientationTargetToSource are used to convert orientation information. The method convertOrientation converts a specified Orientation with respect to a local tangent frame associated with a specified origin point in the source coordinate reference frame, to the local tangent frame associated with the specified origin point in the target coordinate reference frame. The method convertOrientationTargetToSource reverses the roles of the source and target coordinate reference frames.

The methods convertVector, convertVectorTargetToSource, and convertVectorInBodyFrame are used to convert vector quantities, such as velocity and acceleration information. The method convertVector convert a specified vector quantity with respect to a local tangent frame associated with a specified origin point in the source coordinate reference system, to the local tangent frame associated with a specified origin point in the target coordinate reference frame. The method convertVectorTargetToSource reverses the roles of the source and target coordinate reference frames. The method convertVectorInBodyFrame allows the source vector to be specified in terms of a body frame rather than a local tangent frame, by specifying the orientation of the body frame with respect to a local tangent frame.

The class Orientation represents the orientation of one set of 3D coordinate reference frame axes with respect to another. It provides a constructor, as well as five pairs of access methods that support five different orientation representations:

1) Orientation, which initializes the object to an identity matrix,
2) setAxisAngle and getAxisAngle,
3) setEulerAnglesZXZ and getEulerAnglesZXZ,
4) setTaitBryanAngles and getTaitBryanAngles,
5) setMatrix3x3 and getMatrix3x3, and
6) setQuaternion and getQuaternion.

These five orientation parameter classes represent various ways of representing orientation information.

The axis-angle representation of an orientation consists of a unit vector $\boldsymbol{n}$ (with components n_1, n_2, and n_3) and a rotation angle θ. As shown in Figure 25, this represents a rotation of the world coordinate axes through the angle θ about the axis defined by $\boldsymbol{n}$. This rotation, indicated by the red arrows, relates the world coordinate axes (x, y, z), shown in green, with the aircraft body axes (x', y', z'), shown in blue. The green-tinted plane in the figure is parallel to the xy-plane of the world coordinate system, while the blue-tinted plane is the aircraft body $x'y'$-plane. The rotation direction is determined by the right hand rule, i.e., if the right hand grasps the vector, with the thumb pointing in the direction of the vector, the fingers curl around the vector in the direction of the rotation angle θ.

Figure 24: JView Orientation and Vector Conversion Service API

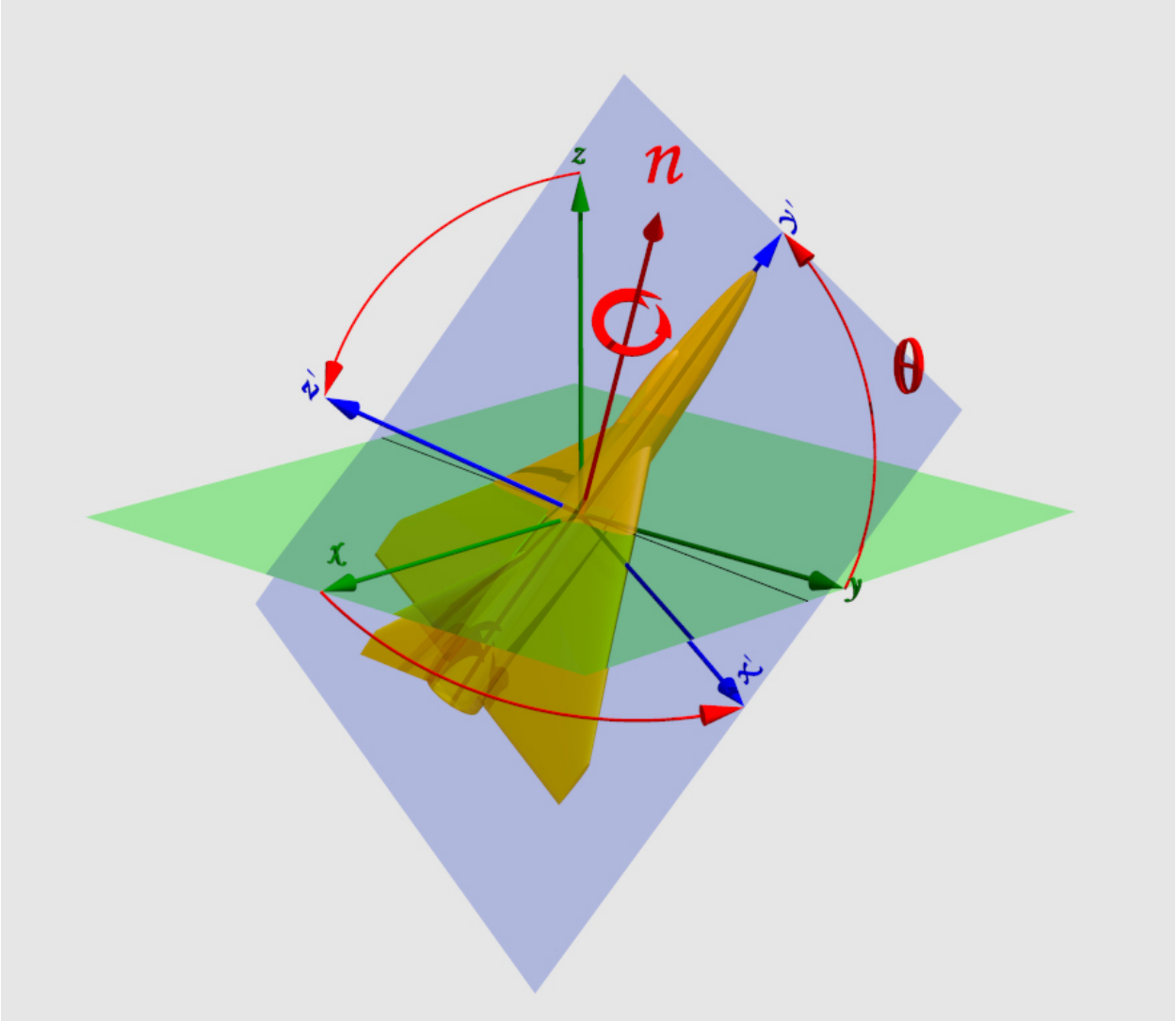

Figure 25: Axis-Angle Representation of Orientation

The class AxisAngle specifies the parameters that allow an Orientation object to be instantiated using an axis-angle representation. It consists of an axis, specified by a 3D vector, and a rotation angle about that axis. The vector is expressed in terms of its three components in the world CRF. The rotation angle is specified in radians.

Euler angles specify an orientation in terms of three consecutive rotations about the principal coordinate system axes. There are twelve distinct ways to select such a sequence of rotations (for right-handed axes). Each of these orderings is called an Euler angle convention. Unfortunately, there is little agreement on how to identify these conventions.

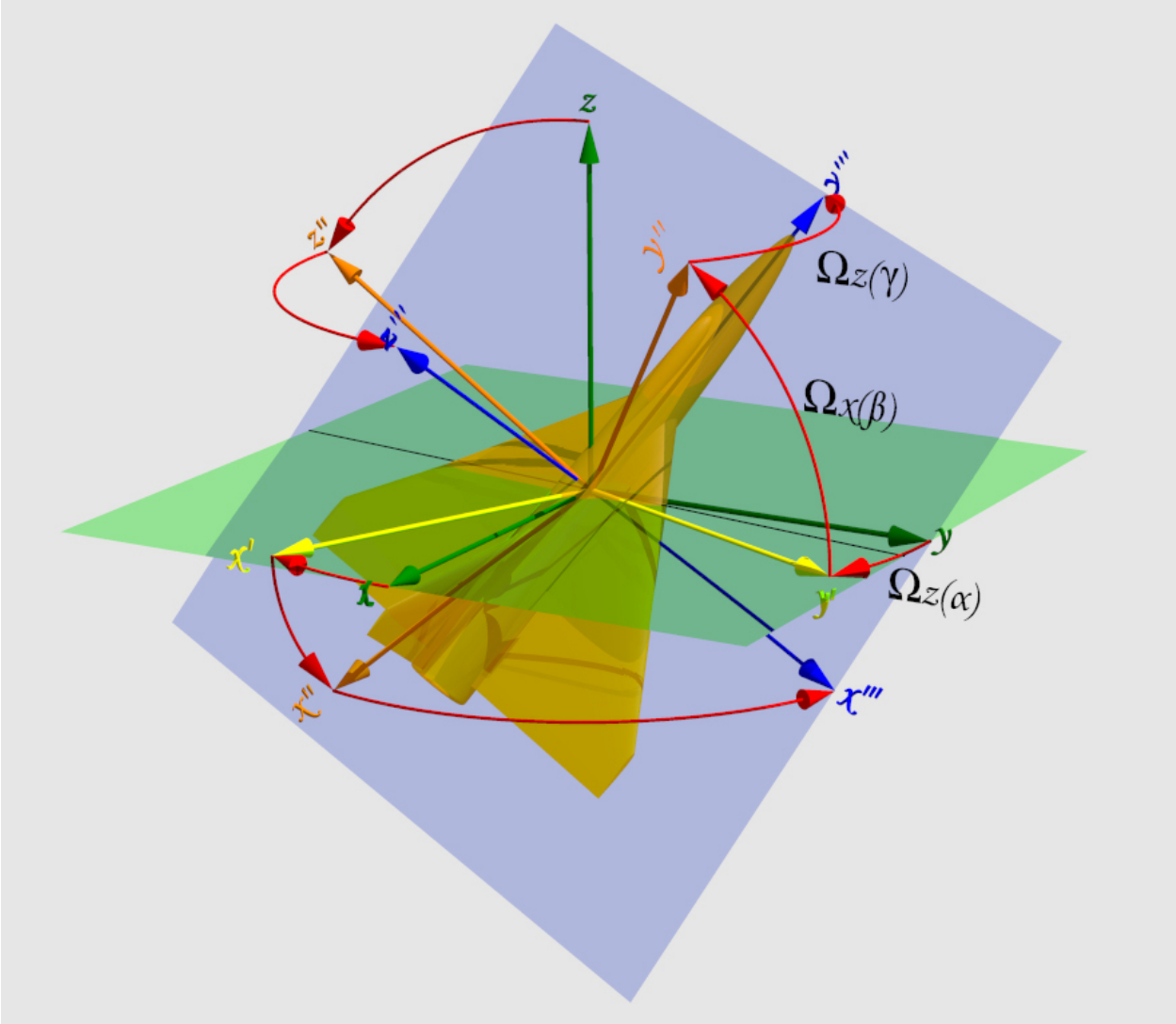

Figure 26: Euler Angle Z-X-Z Representation of Orientation

The JView OVCS software supports the Euler angle convention identified as the ***z-x-z*** convention. This is also known as the 3-1-3 convention, or the ***x***-convention. (The OVCS software also supports the Tait-Bryan angle representation, which is another widely used Euler angle convention.) As shown in Figure 26, this involves a sequence of three rotations that relate the world coordinate axes, shown in green, with the aircraft body coordinate axes, shown in blue. The green tinted plane in the figure is parallel to the world reference system *xy*-plane, while the blue-tinted plane is the aircraft body *x'''y'''*-plane.

The first rotation, $\hat{\mathbf{o}}_z(\alpha)$, is about the *z*-axis, through angle α. This yields the *x'* and *y'* axes, shown in yellow in Figure 26, while the *z* axis, shown in green, remains unchanged.

The second rotation, $\hat{\mathbf{o}}_x(\beta)$, is about the (original) *x*-axis, through angle β. This yields the *x''*, *y''*, and *z''* axes, shown in orange in Figure 26.

The third rotation, $\hat{\mathbf{o}}_z(\gamma)$, is again about the (original) *z*-axis, through angle γ. This yields the *x'''*, *y'''*, and *z'''* axes, shown in blue in Figure 26, which are aligned with the aircraft body axes.

The red arrows in Figure 26 show how the *x*, *y* and *z* axes, shown in green, are progressively transformed by each of these rotations to become first the *x'*, *y'* and *z'* axes, shown in yellow, then the *x''*, *y''* and *z''* axes, shown in orange, and finally the *x'''*, *y'''* and *z'''* axes, shown in blue.

In some contexts, α is called the spin angle, β is called the nutation angle, and γ is called the precession angle.

The class EulerZXZAngles specifies the parameters that allow an Orientation object to be instantiated using an Euler angles ZXZ representation. It consists of three rotation angles specified in radians.

Another widely used Euler angle convention is the ***x-y-z*** convention. This convention is used in the Distributed Interactive Simulation (DIS) standard. Euler angles in this convention are called Tait-Bryan angles. They are also sometimes called Cardano angles, or nautical angles. As shown in Figure 27, this involves a sequence of three rotations that relate the world coordinate axes, shown in green, with the aircraft body coordinate axes, shown in blue. The green tinted plane in the figure is parallel to the world reference system *xy*-plane, while the blue-tinted plane is the aircraft body *x'''y'''*-plane.

The first rotation, $\hat{\mathbf{o}}_x(\varphi)$, is about the *x*-axis, through angle φ. This gives the *y'*, and *z'* axes, shown in yellow in Figure 27, while the *x* axis remains unchanged.

The second rotation, $\hat{\mathbf{o}}_y(\theta)$, is about the (original) *y*-axis, through angle θ. This gives the *x''*, *y''*, and *z''* axes, shown in orange in Figure 27.

The third rotation, $\hat{\mathbf{o}}_z(\psi)$, is about the (original) *z*-axis, through angle ψ. This gives the *x'''*, *y'''*, and *z'''* axes, shown in blue in Figure 27, which are aligned with the aircraft body axes.

The red arrows in Figure 27 show how the *x*, *y*, and *z* axes, shown in green, are progressively transformed by each of these rotations to become first the *x'*, *y'*, and *z'* axes, shown in yellow, then the *x''*, *y''*, and *z''* axis, shown in orange, and finally the *x'''*, *y'''*, and *z'''* axis, shown in blue.

In some contexts, φ is called the roll (or bank or tilt) angle, θ is called the pitch angle, and ψ is called the yaw (or heading or azimuth) angle. Figure 27 is consistent with these terms as used with an East-North-Up (ENU) axis convention.

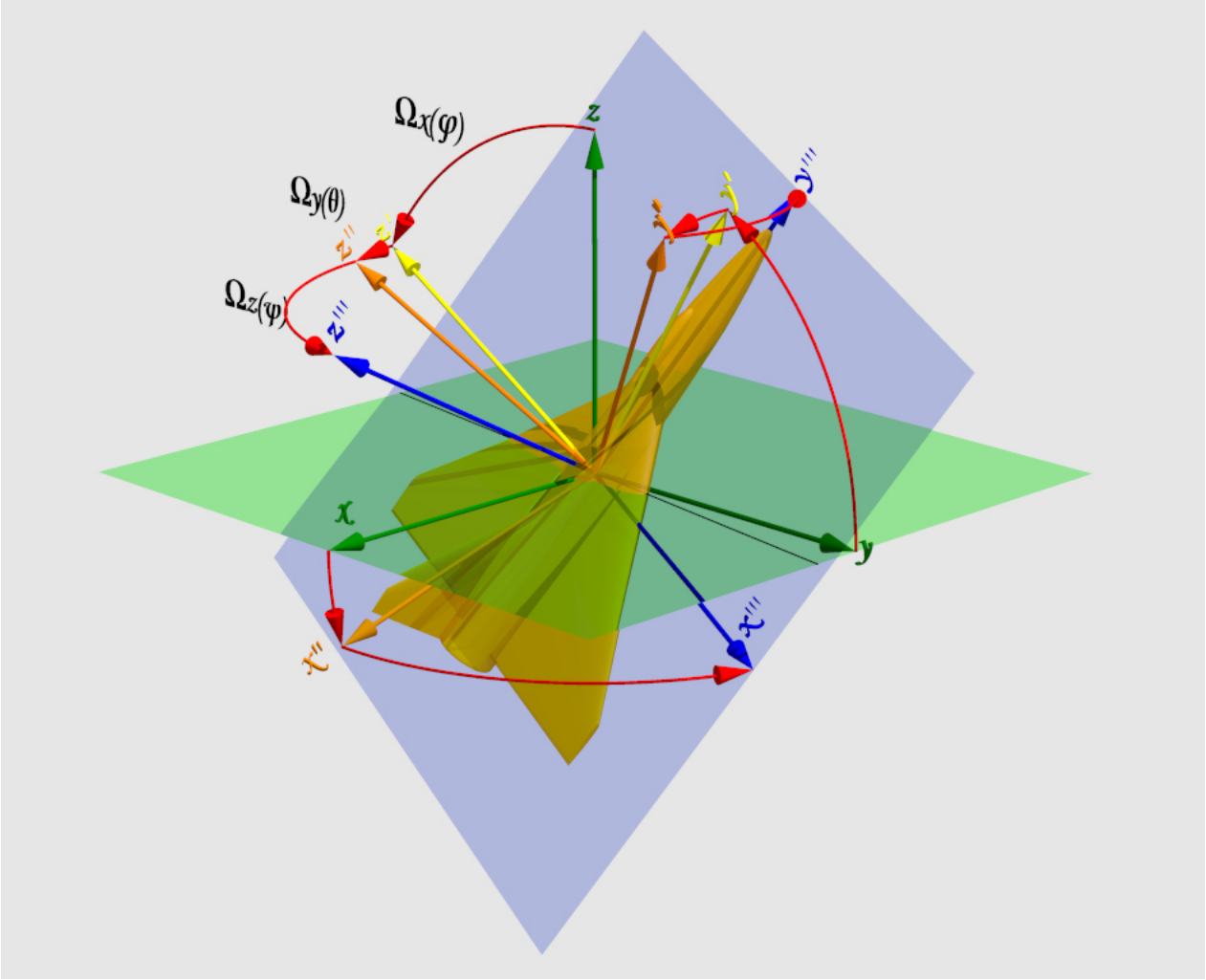

Figure 27: Tait-Bryan Angle Representation of Orientation

The class TaitBryanAngles specifies the parameters that allow an Orientation object to be instantiated using a Tait-Bryan angles representation. It consists of three rotation angles specified in radians.

The orientation of a rigid body can also be represented in the form of a 3x3 rotation matrix. To represent the Euler angle ***z-x-z*** convention discussed above, the equivalent 3x3 rotation matrix representation is:

$$\boldsymbol{\Omega}_{\mathbf{z}}(\alpha)\,\boldsymbol{\Omega}_{\mathbf{x}}(\beta)\,\boldsymbol{\Omega}_{\mathbf{z}}(\gamma) =$$

$$\begin{pmatrix} \cos\alpha\cos\gamma - \cos\beta\sin\alpha\sin\gamma & \cos\beta\sin\alpha\cos\gamma + \cos\alpha\sin\gamma & \sin\beta\sin\alpha \\ -\sin\alpha\cos\gamma - \cos\beta\cos\alpha\sin\gamma & \cos\beta\cos\alpha\cos\gamma - \sin\alpha\sin\gamma & \sin\beta\cos\alpha \\ \sin\beta\sin\gamma & -\sin\beta\cos\gamma & \cos\beta \end{pmatrix}$$

The class Matrix3x3 specifies the parameters that allow an Orientation object to be instantiated using a 3x3 rotation matrix representation. It consists of the nine matrix elements.

The word "quaternion" means "a set of four". Quaternions are elements of a 4-dimensional vector space. They were first described by the Irish mathematician Sir William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space. From a purely geometric point of view, a quaternion may be regarded as the quotient of two vectors, or, equivalently, as the operator that transforms one vector into another. Due to certain compactness, efficiency, and stability advantages over matrices, quaternions have found their way into applications in computer graphics, robotics, global navigation, and the orbital mechanics of satellites.

In analogy to complex numbers, quaternion axes $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$, are defined with the following relationships: $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$. A quaternion $\boldsymbol{q}$ is denoted as $\boldsymbol{q} = e_0 + e_1\boldsymbol{i} + e_2\boldsymbol{j} + e_3\boldsymbol{k}$. This is known as the Hamilton form. The first term e_0 is called the "real" (or "scalar") part of $\boldsymbol{q}$ and $e_1\boldsymbol{i} + e_2\boldsymbol{j} + e_3\boldsymbol{k}$ is called the "imaginary" (or "vector") part of $\boldsymbol{q}$. The OVCS software uses a convention known as the 4-tuple form, which is simply the 4-tuple of numbers $\boldsymbol{q} = (e_0, e_1, e_2, e_3)$.

The class `Quaternion` specifies the parameters that allow an Orientation object to be instantiated using a quaternion representation. It consists of a 4-tuple of numbers, the scalar part and the three vector parts. The parameter values must meet the constraint: $e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1$

Finally, the class Vector3D represents a vector quantity in a 3D coordinate reference frame. It provides a constructor, as well as three pairs of access methods for the individual vector components:

The JView Orientation and Vector Conversion application is called JView OVCS GEOTRANS 3.0J. It is an extended and customized version of the JView GEOTRANS 3.0J application. Its GUI is shown in Figure 28. The left side of the GUI is a customized version of the JView GEOTRANS 3.0J GUI for coordinate conversion that is limited to three-dimensional coordinate system types, i.e., Geodetic, Geocentric, and Local Cartesian. It is used to specify the source and target coordinate systems and datums for orientation and vector conversion operations. It can also be used to perform coordinate conversions among the three-dimensional coordinate system types. The right side of the GUI supports orientation and vector conversion operations between the three-dimensional coordinate system types selected on the left side. The Options menu is used to select between orientation, vector, and vector in body frame conversion operations.

For orientation conversion operations, a pull-down menu on the right side allows any of the supported orientation representations to be selected, causing appropriate fields to be displayed. For vector conversion operations, the three vector component fields are displayed. For both orientation and vector conversion operations, the conversion buttons between the upper and lower panels can be used to convert in either direction. For vector in body frame conversion operations, which are not bidirectional, the upper panel displays both orientation and vector component fields for input, while the lower panel displays the vector componet outputs.

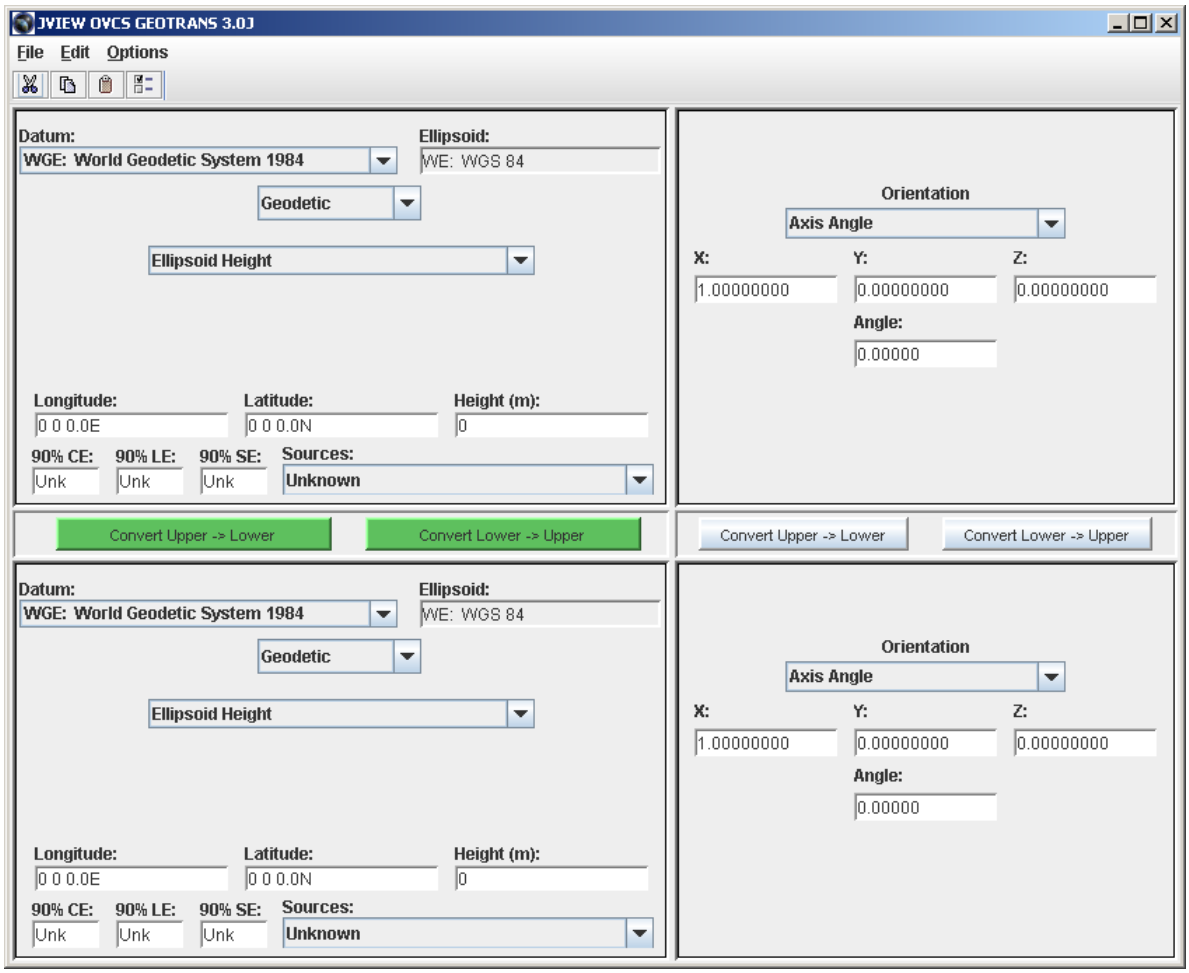

Figure 28: JVIEW OVCS GEOTRANS 3.0J Application GUI

Testing of the JView Orientation and Vector Conversion Service software was performed using the set of test cases being developed by the SEDRIS organization for testing the orientation and vector conversion operations that are being added to the SEDRIS SRM. However, this set of test cases is still under development, and was very limited at the time that the testing of the JView Orientation and Vector Conversion Service software was performed. Further testing of the JView Orientation and Vector Conversion Service software is recommended once the SEDRIS SRM orientation and vector conversion test set becomes fully mature.

4 DISTRIBUTED MISSION OPERATIONS (DMO) TEST HARNESS (DMOTH)

The Distributed Missions Operations (DMO) Test Harness (DMOTH) software was developed to support research and experimentation on DMO Infrastructure technologies. It is composed of freely releasable open source and government software components. As shown in Figure 29, it includes a set of simple applications that send and receive distributed simulation traffic using the Distributed Interactive Simulation (DIS) and High Level Architecture (HLA) protocols. Other existing DMO applications can also be included, though it may be difficult to precisely control the amount and types of message traffic that they generate. A control application allows multiple sender and receiver applications to be started, paused, and stopped on multiple networked systems running various Windows or Linux operating systems. All sent and received data packets are archived in local MySQL databases. XML configuration files are used to control the numbers and locations of the sender and receiver applications, as well as the distributed simulation traffic generated by each sender application. A post-processing application then gathers all of the archived data into a single database, relating the sent and received data packets to each other. This database can then be queried to evaluate packet loss, packet latency, and a variety of other performance criteria.

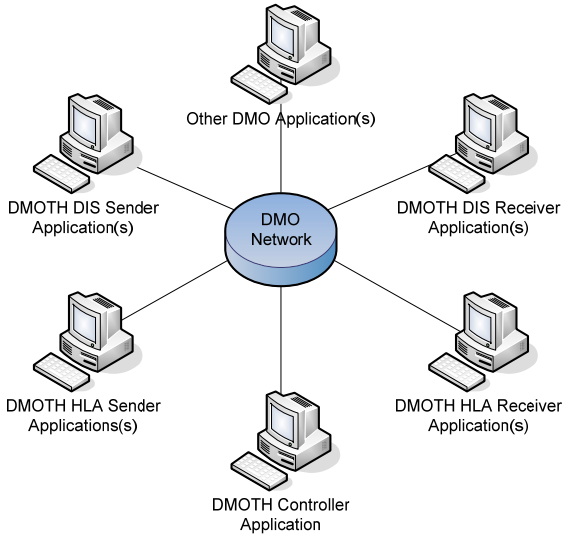

Figure 29: DMOTH System Configuration

The architecture of the DMOTH software is shown in Figure 30. It consists of four types of applications:

1) DMOTH Sender Application – which generates, sends, and archives distributed simulation data packets using either the DIS or HLA protocol,

2) DMOTH Receiver Application – which receives and archives distributed simulation data packets using either the DIS or HLA protocol,

3) DMOTH Controller Application – which allows multiple DMOTH Sender and Receiver Applications to be started and controlled on multiple networked computers, and

4) DMOTH Analyzer Application – which integrates the individual databases created by the DMOTH Sender and Receiver applications to archive the sent and received data packets into a single database for each experiment run that can be used to analyze the performance of the DMO network.

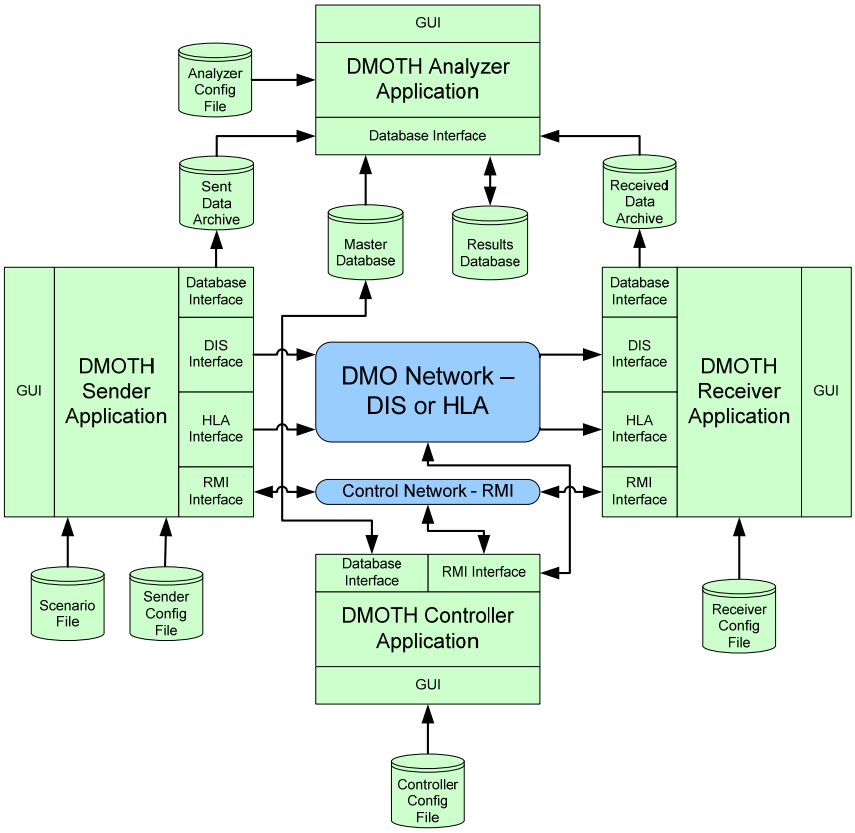

Figure 30: DMOTH Software Architecture

The DMOTH applications communicate with each other using two networks: the DMO Network supports either the DIS or HLA protocols to send and receive distributed simulation data packets. A separate control network supports communication between the DMOTH Controller application and multiple DMOTH Sender and Receiver applications, using the Java™ Remote Method Invocation (RMI) mechanism. Each DMOTH Sender and Receiver application registers with the local RMI server on the computer where it is executing. The DMOTH Controller connects with each Sender and Receiver application, and then passes them instructions that specify their role in the experiment run.

Normally, the DMOTH Controller application runs on one computer, while multiple DMOTH Sender and Receiver applications run on other computers on the network. Although multiple DMOTH Sender and/or Receiver applications can be run on the same computer, normally each runs on a separate computer, so that they do not compete with one another for resources.

The DMOTH Controller application reads an XML configuration file that specifies how many Sender and Receiver applications are to be run, which computer each application is to run on, and where each application is to archive the data packets that it sends or receives. It also contains the RMI connection information needed to establish communication between the DMOTH Controller application and each Sender and Receiver applications.

The DMOTH Controller application creates and manages a "master" database (dmoth_tables) that keeps track of the data packets archived during each experiment run. It consists of a single table that contains an entry for each data packet archive created by the DMOTH Sender and Receiver applications during each experiment run. This information is used by the DMOTH Analyzer application to find and integrate the individual archives.

Each Sender or Receiver application reads a simpler XML configuration file that contains corresponding RMI connection information. Once the RMI connection is established, each application receives information from the DMOTH Controller, specifying what distributed simulation protocol (DIS or HLA) is to be used, and where it is to archive the data packets that it sends and receives. Each DMOTH Sender application also reads an XML scenario file, specified by the Controller, which specifies what type(s) and amount of simulation data packets are to be generated.

The DMOTH Analyzer application is a post-processing application that reads the "master" database created by the DMOTH Controller application, and uses the information that it contains for a specified experiment run to integrate the individual data packet archives created by each DMOTH Sender and Receiver application into a results database for that experiment run. This results database contains a table that contains all of the data packets sent by all DMOTH Sender applications during that experiment run, another table that contains all of the data packets received by all DMOTH Receiver applications during that experiment run, and a join table that associates each data packet that was sent with each of the corresponding data packets that were received. The results database can then be analyzed to determine how many packets were lost, the latency of each packet, etc.

DMOTH is a distributed software system designed to operate on Intel x86 compatible hardware running Microsoft Windows and/or Linux operating systems. The disk drive(s) used by the DMOTH software must be large enough to store the sent and received data packets that the DMOTH archives, as well as the analysis database, which associates sent packets with received packets. The recommended hardware requirements are as follows:

Processor:	Intel Pentium 4 3.0 GHz or equivalent
Memory:	2GB minimum
Storage:	60GB minimum
Optical Drive:	CD±R or DVD±R drive

The prerequisite software, which must be installed prior to installing the DMOTH software, includes:

1) Sun Java™ SE Run-time Environment (JRE), version 1.6 or newer,
2) MySQL.Server, version 5.1 or newer, and
3) MySQL Tools, version 5.0 or newer.

MySQL is a relational database management system (RDBMS), which runs as a server providing multi-user access to a number of databases. MySQL is owned and sponsored by the Swedusg company MySQL AB, which is now a subsidiary of Sun Microsystems, which holds the copyright to most of the codebase. The source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements. Downloads are available at: http://www.mysql.com/.

The DMOTH software depends on several external components. These external components include:

1) Open-DIS – an open source implementation of the Distributed Interactive Simulation (DIS) standard;
2) Portico – an open source implementation of the High Level Architecture (HLA) Run Time Infrstructure (RTI) for distributed simulation;
3) Jpcap – an open source Java™ library for capturing and sending network packets;
4) Groovy – an open source, Java-based scripting language (version 1.6 or newer).

Open-DIS is a free, open source implementation of the Distributed Interactive Simulation (DIS) standard (IEEE 1278) in C++ and Java™. Open-DIS was developed primarily by the MOVES Institute at the Naval Postgraduate School. It is available under a BSD licence from: http://open-dis.sourceforge.net/Open-DIS.html.

Portico is an open source, cross-platform High Level Architecture (HLA) Run Time Infrastructure (RTI) implementation that is intended to support production and continued research and development in distributed simulation. Portico is licensed under the terms of the Common Developer and Distribution License (CDDL). Copies of the Portico software can be downloaded at: http://porticoproject.org/index.php?title=Main_Page.

Jpcap is a Java™ library for capturing and sending network packets. Jpcap can capture Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets. Jpcap is open source, and is licensed under GNU LGPL. It has been tested on Microsoft Windows (98/2000/XP/Vista), Linux (Fedora, Mandriva, Ubuntu), Mac OS X (Darwin), FreeBSD, and Solaris. Jpcap is available at http://netresearch.ics.uci.edu/kfujii/jpcap/doc/.

Groovy is used to implement the DMOTH Analyzer application. Groovy is an open source dynamic scripting language for the Java Virtual Machine. It is consistent with Java syntax, and builds upon the strengths of Java, but has additional features inspired by languages like Python, Ruby, and Smalltalk. Groovy compiles to Java bytecode. Groovy uses a BSD / Apache style licence. It is available at http://groovy.codehaus.org/.

The DMOTH software uses several different types of configuration files to control its operation. Each individual DMOTH Sender or Receiver application uses a small XML configuration file to register itself with the local RMI registry, so that when the Controller application initializes, it can communicate with that Sender or Receiver application via RMI. These files are identical for both Sender applications and Receiver applications.

The DMOTH Controller application users a larger XML configuration file to connect to one or more Sender applications and one or more Receiver applications, distributed across multiple systems in a network.

Each individual DMOTH Sender application uses an XML configuration file to control the types and numbers of data packets that it generates.

The DMOTH Analyzer application also uses an XML configuration file to locate the master experiment database created by the Controller application, and to determine where to put the results database containing the results of a specific experiment.

When using the HLA protocol, the DMOTH software requires an HLA Federation Object Model (FOM) file. This file defines the objects and interactions that can be sent and received by the individual federate that make up the federation. The FOM that is to be used by each DMOTH Sender application is specified in the APPLICATION element for that application in the Controller configuration file. For development and testing, DMOTH used the Real-time Platform Reference (RPR) FOM exclusively, primarily because of its compatibility with the DIS protocol. The use of a significantly different FOM might require changes to the DMOTH Sender application software.

When using the HLA protocol, the DMOTH software requires an HLA RTI Initialization Data (RID) file. Because it uses the Portico implementation of the HLA RTI, the DMOTH software uses a Portico RID file.

When using the DMOTH software to run a DMO experiment, the general sequence of steps is as follows:

1) Start each of the individual DMOTH Sender and Receiver applications required for the specific experiment. Normally, each of these applications is run on a separate computer. The user interface of a DMOTH Sender/Receiver application is shown in Figure 31.

2) Start the DMOTH Controller application, and use it to initialize and start the remote Sender and Receiver applications to generate, send, receive, and archive distributed simulation data packets. When a sufficient number of data packets have been sent, received, and archived, use the DMOTH Controller to stop the experiment. The user interface of the DMOTH Controller application is shown in Figure 32.

3) Run the DMOTH Analyzer application to create an integrated database of experiment results, relating the data packets sent by each Sender application to the corresponding data packets received by each Receiver application.

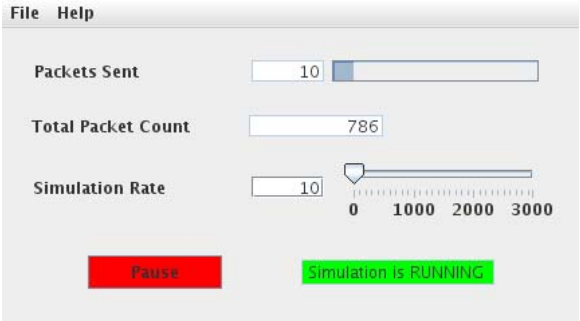

Figure 31:DMOTH Sender/Receiver Application Running

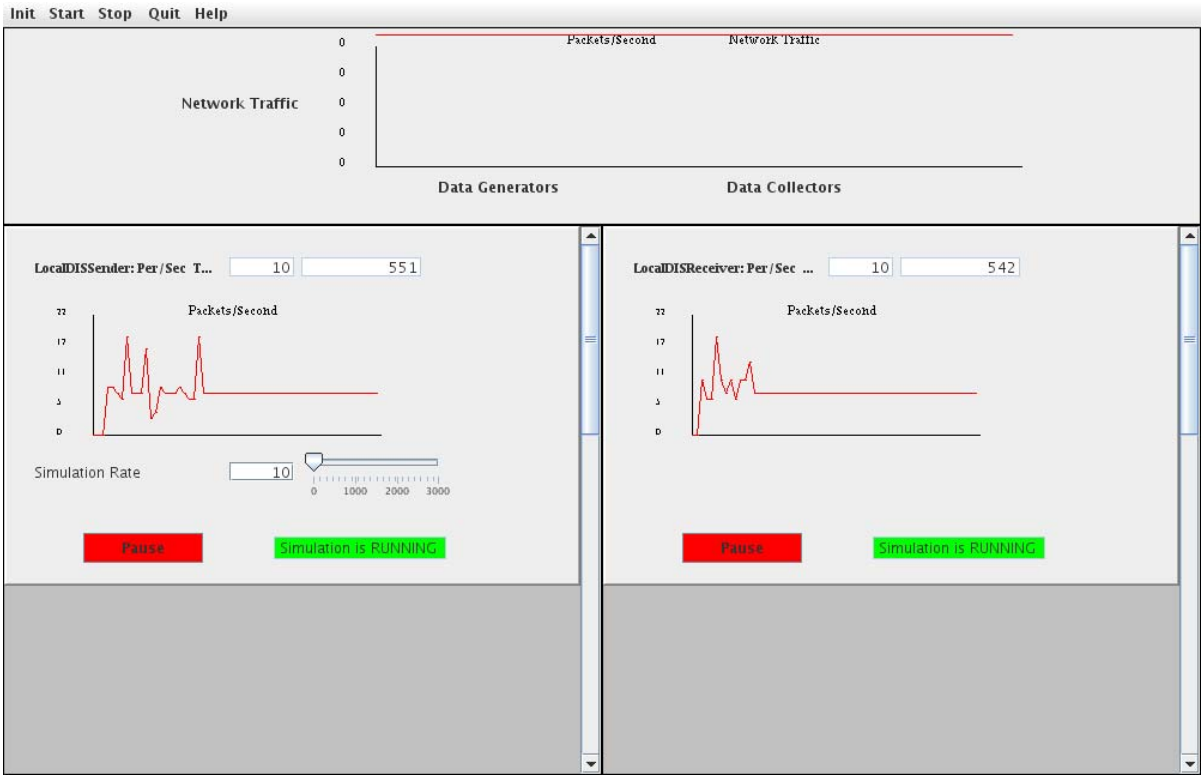

Figure 32: DMOTH Controller Application Running

The DMOTH Analyzer reads the master database created by the Controller application, and creates a results database that integrates the individual databases created by each of the remote Sender and Receiver applications. The results database consists of a table containing all of the data packets sent during the experiment run, another table containing all of the data packets received during the simulation run, and a join table that associates each sent packet with all of the corresponding received packets. This database can then be queried to analyze the performance of the tested configuration.

The MySQL Query Browser, or any other database visualization tool, can be used to view and query the results database, which is named “dmoth_results” concatenated with the name of the experiment run. Figure 33 shows the content of the master_send table, which contains all of the data packets sent by all DMOTH Sender applications during the experiment run. The master_recv table similarly contains all of the data packets received by all DMOTH Receiver applications during the experiment run. The send_recv_join table associates each data packet send with all corresponding data packets received.

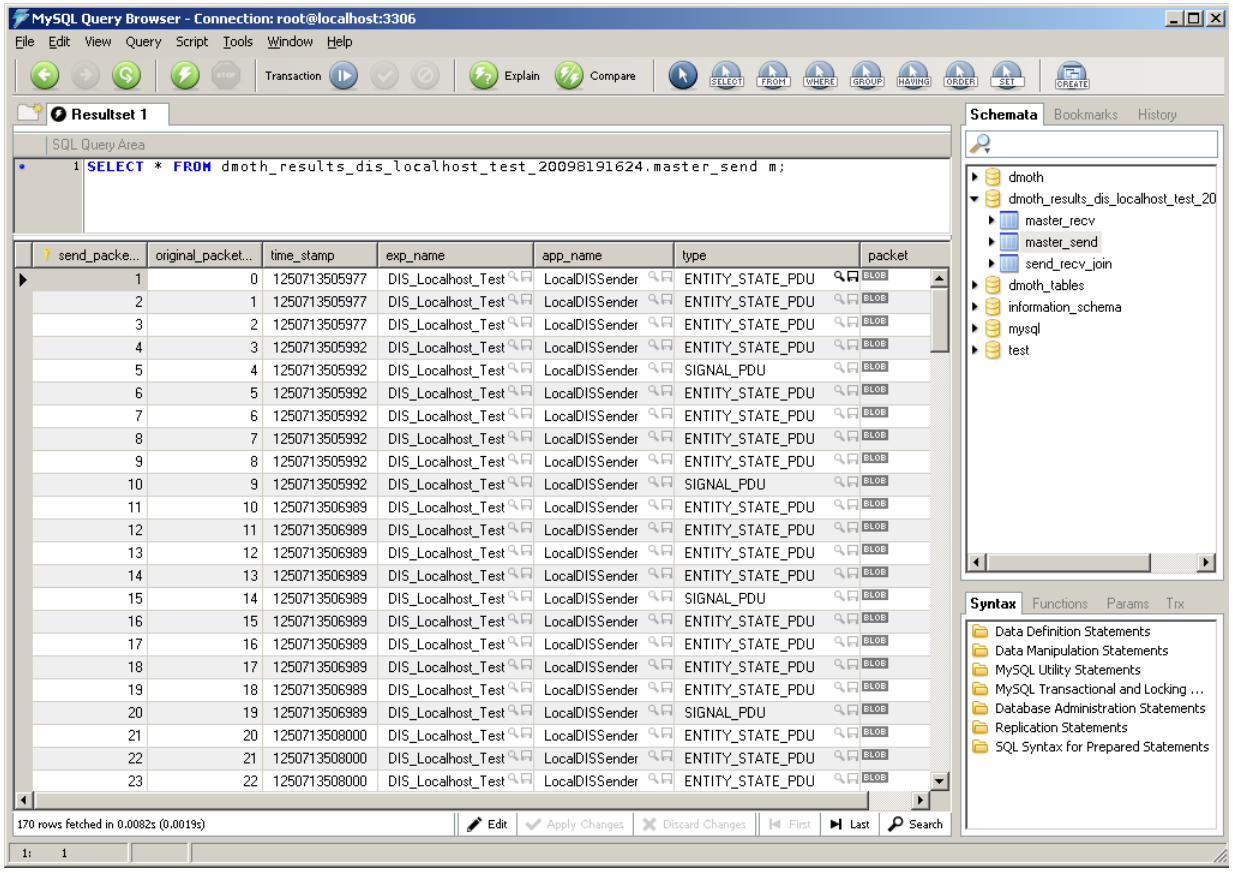

Figure 33: Results Database, Master Sent Packets Table

The DMOTH “master” database is created and managed by the DMOTH Controller Application, and keeps track of multiple experiment runs, and the multiple data packet archives created by each DMOTH Sender and Receiver application that is involved in each simulation run. This database consists of a single table, named “list_of_tables”, which contains one entry for each data packet archive created by DMOTH Sender and Receiver applications.

The DMOTH data packet archive database is created and managed by the DMOTH Controller application, but is populated by the individual DMOTH Sender and Receiver applications that execute during each experiment run. During each experiment run, each DMOTH Sender or Receiver application adds a new table to this database to archive the data packets that it sends or receives. This database thus contains one table for each data packet archive created by a

DMOTH application during a given experiment run. The name of each table is dynamically generated by the DMOTH Controller application, and consists of the name of the experiment run followed by the name of the application that archived the data packets that it contains. Each table contains an entry for each data packet sent or received by a particular application during a particular experiment run, and includes the following columns:

DMOTH experiment run results are created dynamically by the DMOTH Analyzer application. A new experiment run results database is created each time the DMOTH Analyzer application is executed, replacing any existing database with the same name. The name of each DMOTH experiment run results database is "dmoth_results" followed by the name of the experiment run, including the date-time string. An experiment run results database contains three tables:

1) master_send – a table containing all of the data packets sent during the experiment run,
2) master_recv – a table containing all of the data packets received during the experiment run,
3) send_recv_join – a table relating each sent data packet with all of the corresponding received data packets.

The data packet join table (send_recv_join) contains an entry for each data packet sent by one application and received by another. Each entry includes the sent and received packet keys, and the elapsed time in milliseconds, derived from the timestamps in the other two tables.

5 NETWORK-CENTRIC SIMULATION (NSIM)

The Network-centric Simulation (NSim) tool was created by ManTech International Corporation to support the AFRL Network-Centric SIGINT-Focused Information Enterprise (NCSFIE) system. The NCSFIE system is a multi-platform, multi-sensor, distributed intelligent data fusion system of systems. There are many excellent simulation tools available and there was no desire to create a new tool just to test the NCSFIE system. However, after several spiral iterations of the NCSFIE system working with other systems it became very apparent that the available tools were lacking for this application.

The NCSFIE is a data fusion research system that may be configured to ingest a variety of different sensor, network, and database/source feeds. Testing and development using the actual feeds is problematic since this system is not deployed in an operational environment. As a result, the NCSFIE system must generally use simulation data feeds to stimulate the data fusion engine. These simulated feeds may be replaying real-world data or playing synthesized data. One of the key aspects is that the simulation system must implement the data interface(s) of the actual sensor or data network – so as to ensure that the adaptors and logic rules in the NCSFIE system would work with the real sensors/networks when the NCSFIE system migrated to the field.

Existing simulator systems were limited in scope to one type of sensor or data feed, or classified (which greatly hindered distributed team development), or were proprietary (which led to time and cost issues). The NCSFIE team simulation group surveyed a number of simulation tools, standards, and systems. The NCSFIE simulation is dependent on the availability of machine-to-machine interfaces and heterogeneous data types from distributed platforms/sensors/networks. Because these systems were not present, a reasonable match could not be found.

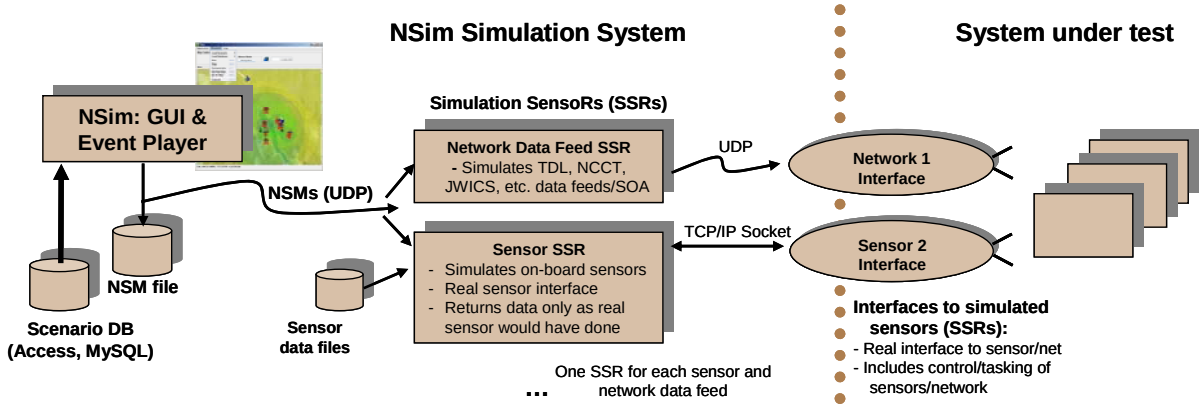

Figure 34: NSim Architecture

NSim was roughly patterned on the Simnet/DIS/PDU entity concept with a loose coupling of the event player and the application-specific client consumers. The NSim application is unclassified and written in Java in order to leverage the available array of enterprise tools and standards. The NSim system architecture is shown in Figure 34.

The primary components of NSim are:

- **GUI/Event Player** The NSim GUI serves two purposes – it provides a visualization of the entities and behaviors during playback, and it is used to edit/create scenarios. The Event Player controls the playback and broadcast of the scenario events. It implements the interface to the database and the messaging control logic. NCSFIE Simulation Messages (NSMs) are output as the simulation "ground truth". The GUI and Event player together comprise the ScenSim application.

- **Scenario Database** The NSim Scenario Database is a MS Access or MySQL database that contains tables which define the simulation entities, their attributes, and their state changes, including Entity State Events (when entities exist), Entity Movement Events (waypoints), and Entity Emission Events (when entities emit signals).

- **Simulation Sensors** The NSim Simulation Sensors (SSRs) simulate the data feeds from sensors, data networks, or other systems into the system under test. The SSRs input the "ground truth" NSMs output by the Event Player. Individual SSRs are designed to implement the actual sensor/network interface messages and protocols of a specific sensor system. SSRs can also accept control/tasking messages from the system under test. They are designed to create and send messages as the actual sensor would when given the tasking commands and ground truth inputs supplied to them.

- **System Under Test Interfaces** The interfaces of the system under test connect to the individual SSRs. Typically these interfaces implement the actual sensor tasking protocols and messages for the sensor system being modeled by an specific SSR.

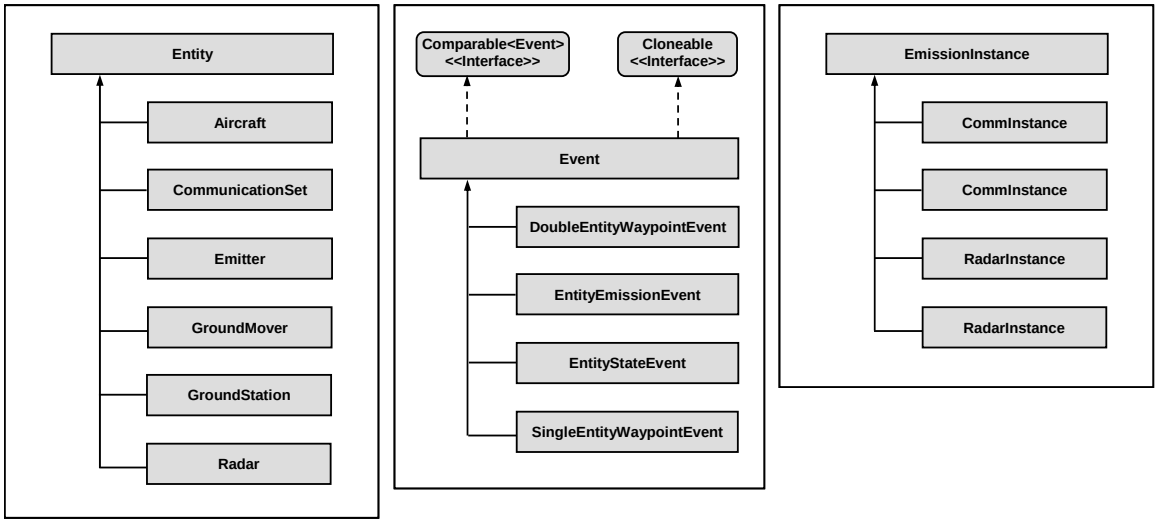

Figure 35: NSim Entity and Event Types

As shown in Figure 35, NSim supports several types of entities, including aircraft, ground vehicles, fixed ground facilities, communication systems, radars, and other emitters. It also supports several types of events, including movement (waypoint) events, emission events, and entity state change events such as creation and destruction. It is important to note that NSim is not really a simulation. It does not actually model the behavior of the entities that it represents, nor are there any causal links between the events that it generates. It is simply a playback mechanism, sorting the entity state changes and events that are stored in its scenario database into temporal order, and output them as messages in the NSM XML-based format.

Under this effort, NSim was interfaced with the JSB-RD distributed simulation environment. This was accomplished using three different methods, as shown in Figure 36:

1) Information was extracted from the TBMCS Air Operations Database (AODB), describing planned friendly air missions, and from the Modernized Integrated Database (MIDB), describing the enemy order of battle, including the military units and facilities that were the targets of the friendly air missions, as well as other relevant military units and facilities, particularly those providing air defense for the air mission targets. This AODB and MIDB information was then used, after some translation, to populate the NSim Scenario Database.

2) The JSAF scenario development tools were used to use the AODB and MIDB information described above to create a JSAF input spreadsheet, mapping the AODB and MIDB entities to the corresponding JSAF entities. This JSAF input data was then used, after some translation, to populate the NSim Scenario Database.

3) JSAF was executed, using the input described above, and the JLogger software was used to capture and archive all of the HLA entity state and interaction messages that were generated by JSAF. These archived messages were then used, after some translation, to populate the NSim Scenario Database.

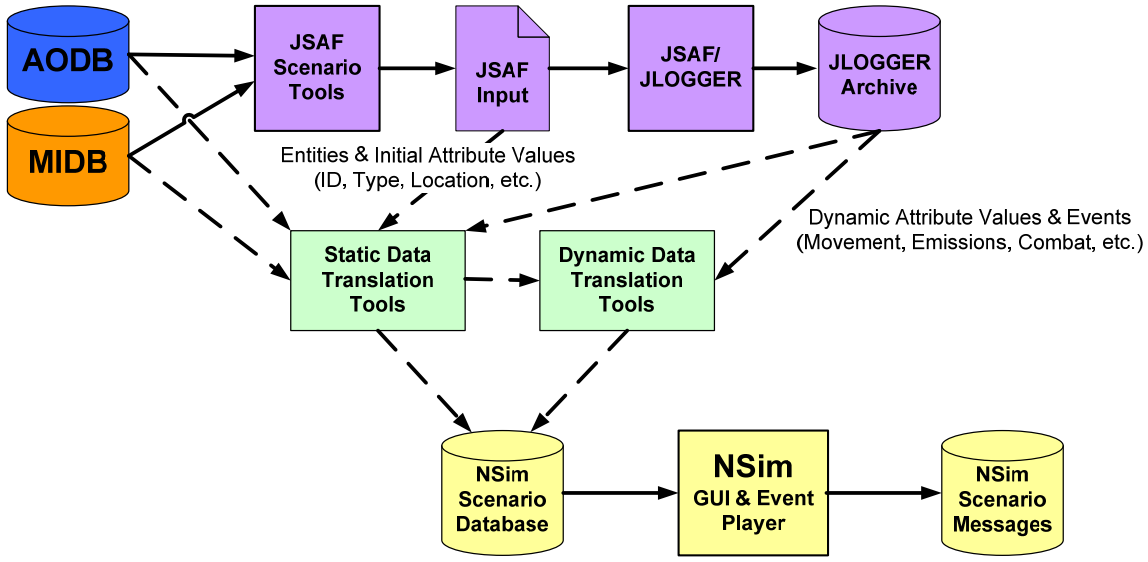

Figure 36: NSim Scenario Database Population Methods

Together, these methods provided both the static data, including entity identifier and type information, initial locations, etc., and the dynamic data, including entity movement, emission, and combat events, needed to populate the NSim Scenario Database. NSim was then executed to play back the scenario.

Table 4 lists the tables that make up the NSim Scenario Database. The contents of each of these tables were mapped to the most appropriate values in the HLA entity state and interaction messages output by JSAF and archived by JLogger. The contents of each of these messages, and the relationships between them, are shown in Figures 37 through 40.

Table 4: NSim Scenario Database Tables

Table Name	Description
ScenarioProperities	Defines Scenario IDs and attributes (names, classifications, initial display views, etc.)
Master	Database info
ScenarioDesignNotes	Design notes
EntityAttributes	Defines all non-changing attributes of the Entity: Unique Entity ID number, Parent ID, name, nomenclature, call sign, country, initial location upon creation.
EntityStateEvents	Defines the time period during which an Entity exists in a scenario.
EntityMovementEvents	Waypoints and times for each Entity for each scenario.
EntityEmissionEvents	On/off times for each Emission event (Radar/Comm set) with the Instance ID (reference to the appropriate Parametric table for all parametric details) for each scenario.
ElintInstanceID Parametrics, ComintInstanceID Parametrics	Define all parametric values for emission events, referenced by InstanceID (from Emission Events). InstanceIDs are unique to an emitter (Entity). They may be reused in multiple events of the same type for the same emitter (EntityID). They may be reused over multiple scenarios.
DisplayAttributes	Defines how to display the entities on the NSim map (visible, emission events, trails, etc.)
EmitterParametricList (EPL)	An ELINT Instance ID parametric comes from an ELNOT with ranges as defined by the EPL.
CommParametricList (CPL)	A COMINT Instance ID parametric comes from a communication set with ranges as defined by the CPL.

Figure 37 shows the abstract BaseEntity and PhysicalEntity classes, as well as the concrete Blip class from the HLA federation object model (FOM) used in the scenario, which is a variation of the Real-time Platform Reference (RPR) FOM. The BaseEntity class provides the identity and type of each entity, along with location, orientation, and velocity information. Acceleration and angular velocity fields are present, but are not populated by JSAF.

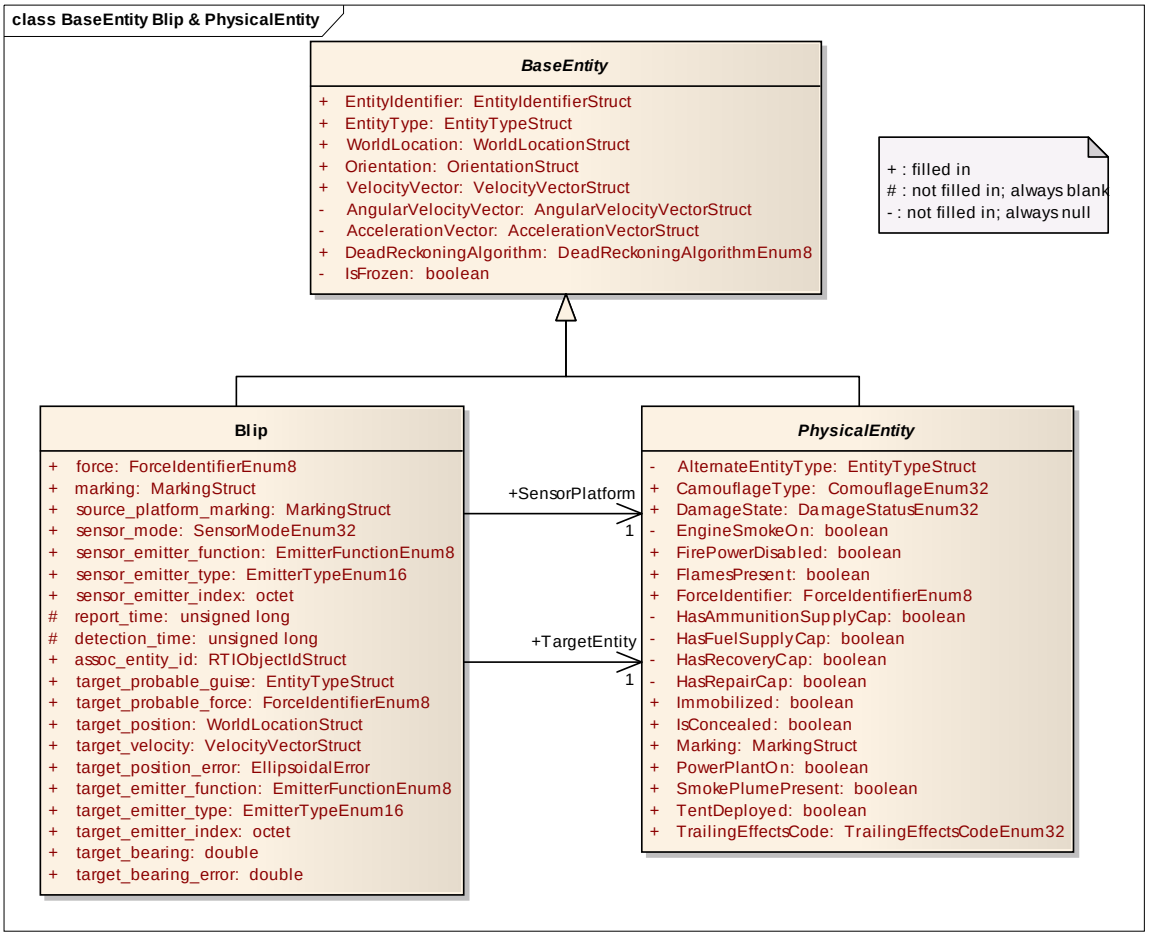

Figure 37: BaseEntity, PhysicalEntity, and Blip Classes

The PhysicalEntity class extends the BaseEntity class, adding damage state information, as well as a variety of appearance flags. Although the PhysicalEntity abstract class has multiple concrete subclasses, including Aircraft, GroundVehicle, and LifeForm, these do not add any additional attribute information, and so are not relevant here.

The Blip class is used to represent detections output by radars and other sensors. The entity that serves as the sensor platform is identified, as is the target entity. Additional information is provided about the sensor that generated the detection. Estimated entity state information, including type, allegiance, location, velocity, and position error are reported. If the target entity is an emitter, emitter function, type, and bearing information is also included.

Figure 38 shows the FOM classes and interactions that are involved in communication events. The RadioTransmitter class represents communications emitters. It provides frequency, bandwidth, power, modulation, antenna, and other information about each radio transmitter object. It also associates each radio transmitter object with the platform entity that carries it. The ApplicationSpecificRadioSignal class provides additional information about individual radio signals, including data rate, message length, and protocol information. Each ApplicationSpecificRadioSignal object is associated with a RadioTransmitter object.

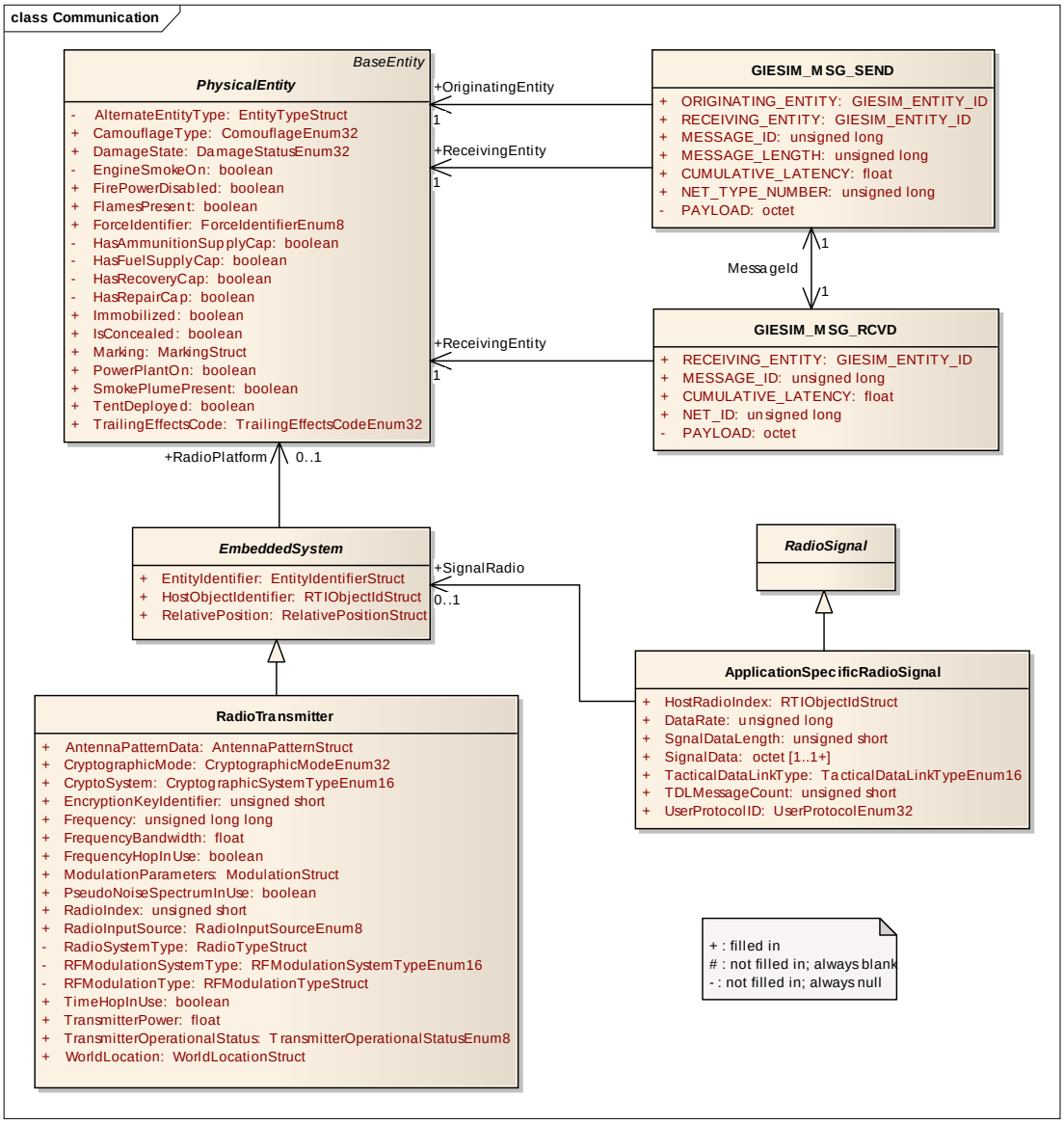

Figure 38: Communication Classes and Interactions

The GIESim software generates two FOM interactions to represent communication events: GIESIM_MSG_SEND and GIESIM_MSG_RCVD. Each GIESIM_MSG_SEND interaction identifies the sending entity, the intended receiving entity, and the unique id of the message, the length of the message, and the cumulative latency of the message. Each GIESIM_MSG_RCVD interaction identifies an entity that received the message, the network id on which the message was received, the cumulative latency of the message, and the unique id of the message. These two interactions types are linked to each other via the unique message id.

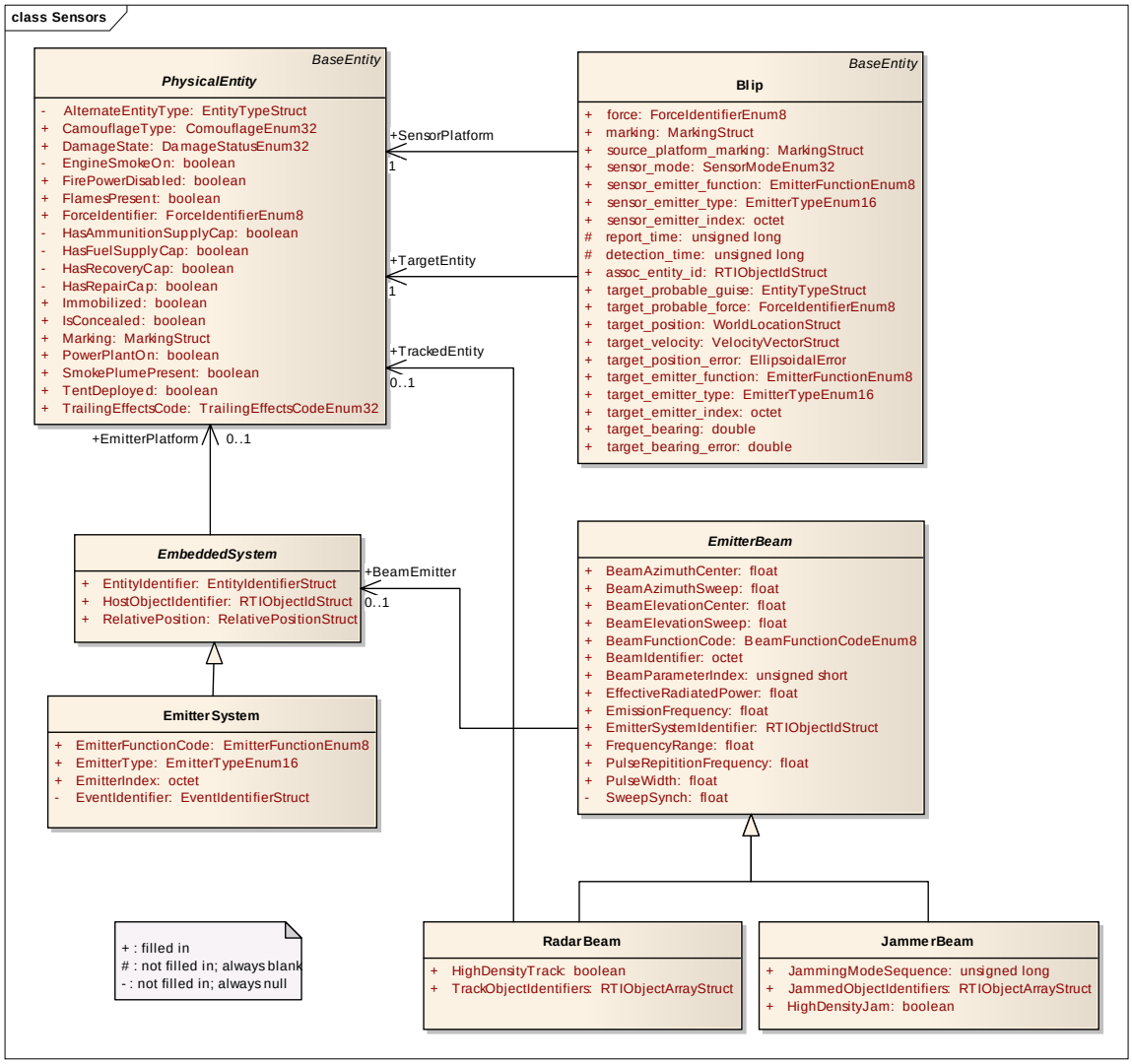

Figure 39: Sensor Classes and Interactions

Figure 39 shows the FOM classes and interactions that are involved in sensor-related events. The Blip class, which represents individual detections, was discussed above. The EmitterSystem class represents radars and other types of active sensors. The abstract EmitterBeam class, and its concrete subclasses RadarBeam and JammerBeam, represent individual beams emitted by radars, other active sensors, and jammers. The abstract EmitterBeam class provides the direction and extent of the beam, as well as parameters such as power, frequency, pulse width, and pulse repetition frequency. Each EmitterBeam is linked to its source emitter, while RadarBeams identify the entity being tracked.

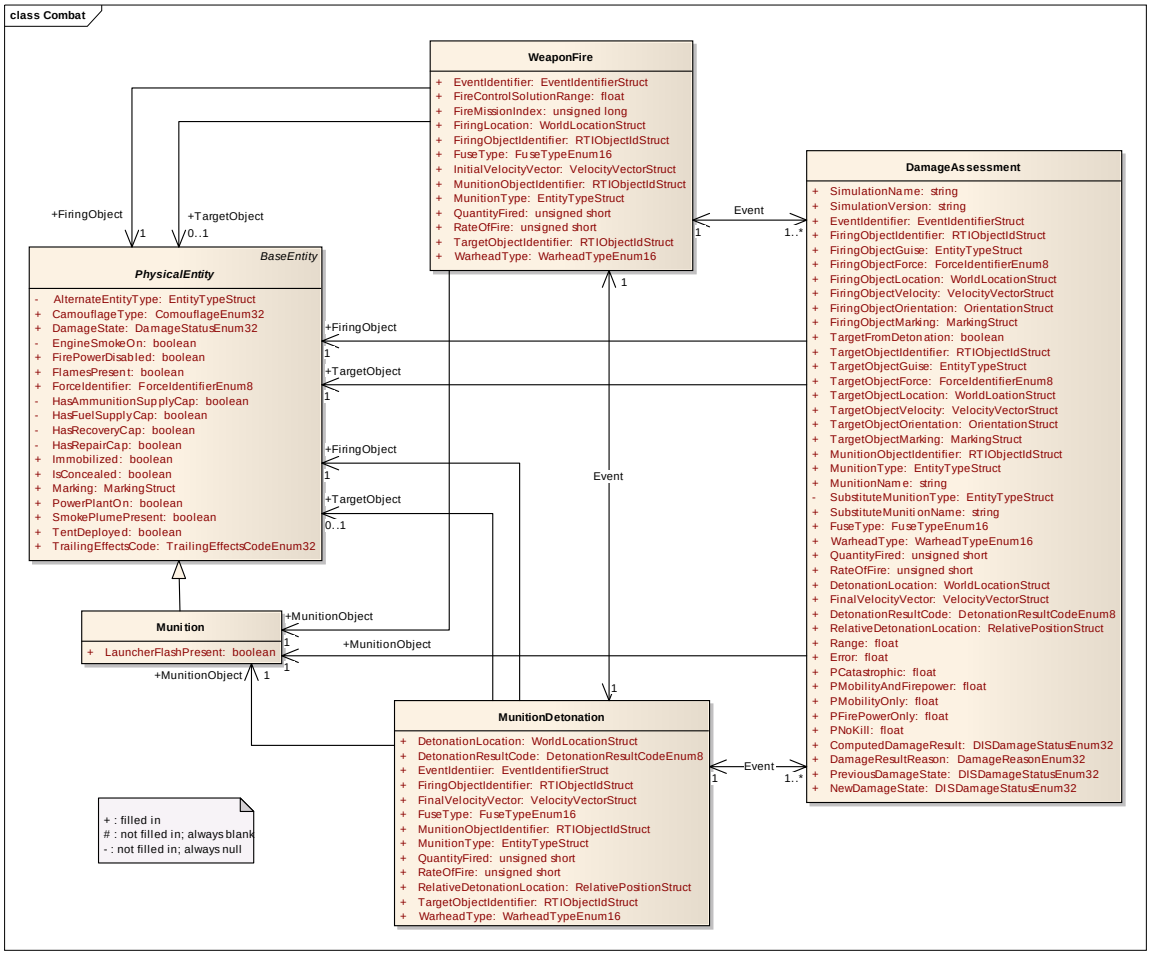

Figure 40: Combat Classes and Interactions

Figure 40 shows the FOM classes and interactions involved in combat events. The WeaponFire interaction represents the firing of a specific weapon. The firing entity is identified, as is the target entity. For missiles and other types of munitions that exist for a significant time while in transit to the target, the munition entity may also be explicitly identified. For guns, the number of rounds fired and the rate of fire are provided, along with warhead and fuse type information, firing location, intended range, and initial velocity vector of the munition.

The MunitionDetonation interaction represents the detonation of the munition. Again, the firing entity, target entity, and munition entity are all identified. Most of the information in the WeaponFire interaction is repeated, except that the detonation location replaces the firing location, and the final velocity vector replaces the initial velocity vector. The MunitionDetonation interaction is linked to the corresponding WeaponFire interaction via the unique event identifier.

The DamageAssessment interaction provides the results of the MunitionDetonation, in terms of the effects on entities near the detonation location. Each DamageAssessment interaction is linked to the corresponding WeaponFire and MunitionDetonation interactions via the unique event id. Each DamageAssessment interaction also identifies the firing entity, the target entity, and the munition entity. However, in this case, the target entity is not necessarily the intended target identified in the WeaponFire and MunitionDetonation interactions. Instead, there is a separate DamageAssessment interaction for each entity that is potentially affected by the detonation. Each DamageAssessment interaction contains entity state information, including location, orientation, and velocity, for both the firing and target entities, as well as warhead and fuse type information, and/or number of rounds and rate of fire information, as appropriate. Most important, each DamageAsssessment interaction contains information on how the detonation event affected a specific target entity, including the range and relative position of the detonation with respect to the target entity, various types of kill probabilities, and, finally, the previous and new damage states of the target entity.

Field Name	Data Type	Description	Source
EntityID	Long Int	Nonzero unique entity ID	JSAF Jlogger (assigned by JSAF)
ParentEntityID	Long Int	EntityID of parent entity, if any	Not Available (Jlogger data field not populated)
EntityType	Text	Class of entity type	Assigned by scenario builder (based on existing scenarios)
EntityNomenclature	Text	Formal designation	JSAF input data (unit type identifier)
EntityName	Text	Descriptive name	AODB/MIDB (assigned callsign, facility table entry)
EntityOperatingCountry	Text	Two-letter country code	AODB/MIDB (Pacifica Codes)
EntityAllegiance	Text	Hostile, Friendly, Unknown	JSAF Input data
EntityCallsign	Text	Short designator (not necessarily unique)	AODB/JSAF input data (JSAF Entity Marking)
EntityDisplayReference	Text	Reference to display attributes	Assigned by scenario builder (inspection of available icons)
Latitude	Double Float	Latitude in decimal degrees of initial location	MIDB/JSAF input data/Jlogger (data conversion needed)
Longitude	Double Float	Longitude in decimal degrees of initial location	MIDB/JSAF input data/Jlogger (data conversion needed)
AltitudeMeters	DoubleFloat	MSL Height of initial location	MIDB/JSAF input data/Jlogger (data conversion needed)

Figure 41: EntityAttribute Table Mapping

Figure 41 summarizes how the NSim Entity Attribute Table is populated, using a combination of information extracted from the MIDB, AODB, and JSAF. MIDB and AODB are used for most static information on adversary and friendly entities, respectively. Location information comes from MIDB, AODB, or JSAF depending on the entity type. The locations of fixed facilities and installations are obtained from MIDB and AODB, for adversary and friendly entities,

respectively, and are then fed into JSAF. The initial locations of moving entities come from JSAF, after JSAF is used to deploy these entities. The scenario builder is responsible for identifying the entity types, and the icons to be used to display the entities. However, these assignments could be automated by constructing a database that maps MIDB, AODB, and JSAF entity types to NSim entity types and icons.

Field Name	Data Type	Description	Source
ESEIndex	Long Int	Unique entity state event identifier	Arbitrary incremented value
ScenarioID	Long Int	Unique scenario identifier	Assigned by scenario builder
EntityID	Long Int	Unique entity identifier	JSAF Jlogger (assigned by JSAF)
State	Text	Entity state (defined, exists, malfunctioning)	Assigned by scenario builder (defaulted to "exists")
StateDescription	Text	Comments on state change	Not Used
CreateAtScenarioStart?	Boolean	Entity exists from scenario start if TRUE	TRUE except for cruise missiles (missile life span from Jlogger data)
ExistsToScenarioEnd?	Boolean	Entity exists until scenario ends if TRUE	TRUE except for cruise missiles (missile life span from Jlogger data)
BeginDate	Text	Date entity comes into existence	Assigned by scenario builder (missile life span from Jlogger data)
BeginTime	Text	Time entity comes into existence	Assigned by scenario builder (missile life span from Jlogger data)
EndDate	Text	Date entity ceases to exist	Assigned by scenario builder (missile life span from Jlogger data)
EndTime	Text	Time entity ceases to exist	Assigned by scenario builder (missile life span from Jlogger data)

Figure 42: Entity State Events Table Mapping

Field Name	Data Type	Description	Source
EMEIndex	Long Int	Unique movement event identifier	Arbitrary incremented value
ScenarioID	Long Int	Unique scenario identifier	Assigned by scenario builder
Entity	Long Int	Unique entity identifier	JSAF Jlogger (assigned by JSAF)
EventDate	Text	Date of movement event	Assigned by scenario builder
EventTime	Text	Time of movement event	JSAF input data (defined by scenario designer)
WayPointDescription	Text	Description of the way point for user reference	Not Used
GeneralDescription	Text	Overall comments on movement event	Not Used
Latitude	Double Float	Latitude in decimal degrees	Jlogger (data conversion needed)
Longitude	Double Float	Longitude in decimal degrees	Jlogger (data conversion needed)
AltitudeMeters	Double Float	MSL height	Jlogger (data conversion needed)
Sub-Scenario Comment	Text	Identifies sub-scenario in which event is used	Assigned by scenario builder

Figure 43: Entity Movement Events Table Mapping

Figure 42 summarizes how the NSim Entity State Events Table is populated. All entities are assumed to exist throughout the scenario, with the exception of cruise missiles. Cruise missile entities are created when they are launched, and are destroyed when they detonate. The cruise missile entity creation and destruction times are extracted from the Jlogger data generated when JSAF executes the scenario.

Figure 43 shows how the NSim Entity Movement Events Table is populated, using the HLA PhysicalEntity updates that are output by JSAF and captured by Jlogger. These are extracted from the Jlogger archive. Whenever a JSAF entity is moving, the PhysicalEntity entity state updates for that entity are converted into corresponding NSim Entity Movement event records. In general, the PhysicalEntity updates generated by JSAF are of much higher frequency than typical NSim movement events, which normally only include waypoints where the entity significantly changes speed or direction. This produces much more detailed and realistic movement. For example, ground vehicles tend to follow roads much more accurately than when using NSim alone. Aircraft maneuvers are also much more realistic, as they are based on the output of JSAF's 6-degree of freedom flight dynamics model. However, only the location information in the PhysicalEntity updates can be exploited, as the NSim Entity Movement Events Table does not include any fields for orientation or velocity information.

Unfortunately, attempts to map JSAF output to the NSim Entity Emission Event Table, and the associated ELINT Instance and COMINT Instance ID Parametrics Tables, were much less successful. There were several reasons for this:

1) Most emitters are "built-in" to the JSAF platform entity specifications, rather than being separate entities in themselves. It is extremely difficult to modify the existing JSAF entity configurations, requiring the editing of multiple text files containing the specifications, and then the rebuilding of the JSAF system to use the modified files.

2) Similarly, the behavior of the JSAF emitters is largely automated. With the exception of the friendly communications modeled by GIESim, JSAF emitters cannot be explicitly turned on and off. Instead, they generate emissions consistent with their higher-level tasking.

3) The JSAF emitter attributes could not be easily mapped to the NSim emitter parameters in a straightforward manner. The level of detail, and the ways in which parameters were represented were too different to be easily resolved.

4) Finally, in the output generated by JSAF, many of the fields needed to populate the NSim ELINT and COMINT Instance ID Parametrics Tables were not populated with values, but instead contained nulls. This reflects the incomplete implementation of many emitters in JSAF.

Despite these limitations, it was possible to use NSim to play back scenarios that had previously been executed by JSAF and archived by Jlogger. Figures 44 and 45 show NSim executing the same scenario discussed earlier in section 2.3. Figure 44 shows the locations of all entities at the start of the scenario. This figure is comparable to Figure 15. The two B-52s and the CALCMs that each B-52 launches are shown in the lower right. The Califon IADS elements are shown in red in the lower-left part of the screen. Blue strike aircraft, ISR aircraft, airborne networking aircraft, and UAVs are shown in blue in the upper left and center areas of the screen. Figure 45 shows a more detailed view of the Santiago Peak SA-5 site, showing the distribution of the TELs, radars, and command and control vehicles at the site. This figure is comparable to Figure 17.

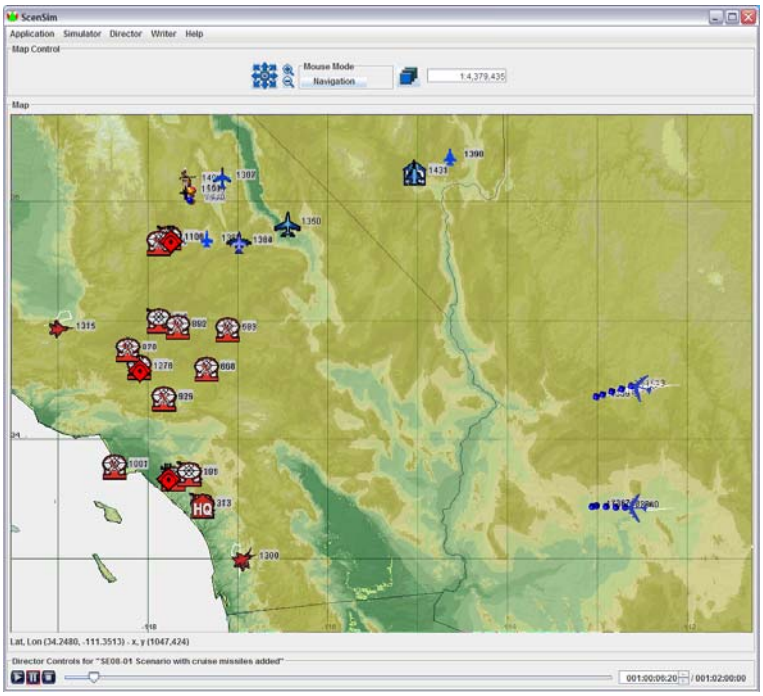

Figure 44: B-52s Launching Cruise Missiles

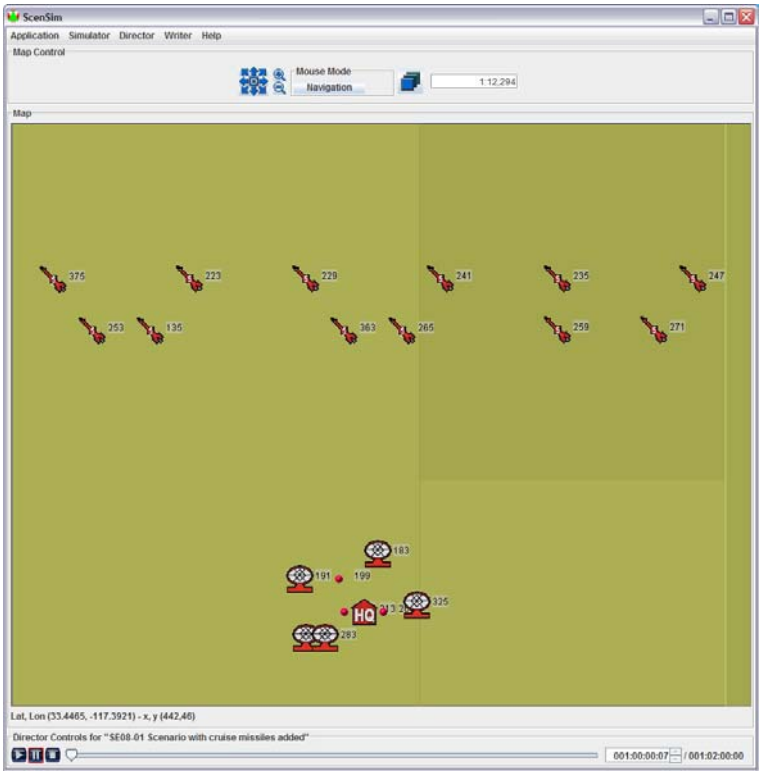

Figure 45: SA-5 Site

6 LESSONS LEARNED, CONCLUSIONS, AND RECOMMENDATIONS

For at least the past 30 years, AFRL/RI (and its predecessor organizations) has needed general-purpose simulation capabilities to support R&D projects in the areas of command and control, sensor exploitation and management, and communications. Prototype C4ISR software tools and systems require realistic input data in order to test and demonstrate their functionality. This data reflects the variety of entities that may populate the battlespace, including adversary and friendly forces, as well as civilian populations. In many cases, this input data also has a significant temporal aspect that reflects the behavior of these entities, as well as the many other events and activities that occur within the battlespace. The importance of joint operations has made it increasingly difficult for the Air Force to focus solely on the air and space domains; the entities that operate in the ground and maritime domains must also be addressed. The increasing importance of the cyber domain has brought a whole new set of challenges, as has the goal of synchronizing air, space, and cyber operations into a single integrated whole.

After working with JSAF for several years now, it has become increasingly clear that JSAF is not the answer to AFRL/RI's simulation needs. The JSAF software contains an impressive set of functionality. However, much of that functionality is very difficult to exploit in practice. JSAF is a very large, very old, and very complex piece of software. JFCOM maintains a staff of developers who are constantly modifying and extending the JSAF software to meet the specific requirements of its ongoing experiments and exercises. Unfortunately, in this process, existing functionality that is not critical to the next JFCOM event is commonly sacrificed. This can leave external JSAF users in a very difficult position. When functionality that AFRL/RI depended on stopped working, it rapidly became impractical to install any further JSAF updates.

Other large DoD simulation systems developed to support training also have a poor track record. For example, JSIMS was cancelled after consuming a very large amount of resources, but before producing any usable results. Under this effort, the Army's OneSAF software was obtained and examined as a possible replacement for JSAF. However, the OneSAF software was so large and so complex, and the documentation so limited, that it was not possible to gain an adequate understanding of how it works, or of whether it might be possible to extend it for use by the Air Force, with the resources available.

Commercial entity-level simulation frameworks, such as MaK Technologies' VR-Forces, or Ternion Corporation's FLexible Analysis, Modeling, and Exercise System (FLAMES), provide a convenient starting point, and a great deal of off-the-shelf functionality. For example, FLAMES has been used successfully by other AFRL directorates, including the Munitions directorate. However, these COTS products tend to be expensive, particularly in the context of small R&D project budgets, or of teams that involve multiple participants from academia and small businesses, as well as government and larger contractors. They also tend to lock users in to a single integrated solution. This has its foundation in the incompatibility of different RTI implementations. Also, in addition to selling these software products, these companies derive a significant portion of their revenue from selling their services as custom simulation developers, as well as training. This creates a conflict of interest with respect to making their software products as easy to understand and use as possible.

Finally, there a non-commercial simulation packages that have been developed to support various programs, including other AFRL programs, such as the JForces packages used in the Airborne Networking Technology (ANT) program. These also contain useful functionality. However, they suffer from some of the same problems as the similar COTS products, including lock-in, and lack of transparency. It is not uncommon for the developers of these systems to go to great lengths to retain effective control of their software, frequently displaying great creativity in making it difficult for others to understand, or even to use, their software without their direct involvement.

Under this effort, the NSim software was evaluated, as described in section 5. While the NSim does have an open structure, a Java implementation, and reasonably clear documentation, it is not actually a simulation, as it does not provide any cause-and-effect linkage between the various kinds of events that are included in its scenarios. It is merely a playback mechanism, which sorts the events defined in its scenario database into the proper order and outputs them in the form of a stream of XML messages. Therefore, while it can be used either as a front end driver to other software tools, including other simulations, or as a back end playback mechanism, as was demonstrated using archived JSAF output, the output of NSim is merely a restatement of its input, with no additional value added. Also, it's lack of support for both the DIS and HLA simulation protocols means that it cannot be easily integrated with other simulation tools.

AFRL/RI needs an open source simulation framework that can be used to support a wide variety of R&D projects across multiple topic areas, including command and control, operational planning, operational assessment, communications networking, and sensor exploitation and management, and which it can freely provide to its contractors and other partners. This effort should be set up and managed as a long-term "meta-project", drawing on multiple projects across AFRL/RI for human and financial resources, but not too dependent on any one project for its survival. It should be possible for individual contributors to come and go as new projects start, and old projects end, or transition to engineering development.

This effort should make the fullest possible use of other open source software projects. For example, it should leverage both the OpenDIS and PoRTIco projects to provide support for the DIS and HLA protocols, respectively. For maximum flexibility, it should be Java-based, and should make use of other open-source technologies such as the MySQL database management system, and open source XML tools.

It should include a simulation engine that is capable of supporting either constrained or unconstrained simulation, including real-time, scaled real-time, or as-fast-as-possible execution. It should support parallel and distributed simulation methods, including both conservative and optimistic synchronization.

It should include a framework of simulation classes representing entities, such as aircraft, facilities, and ground vehicles, aggregates, such as military units, communication networks, and installations, and components, such as sensors, weapons, and communication systems. It must also be able to represent a variety of different types of relationships among instances of these classes, representing command relationships, support relationships, and other types of functional and operational relationships. This simulation framework should be extensible, so that concepts for new systems, including military units (i.e., swarms of UAVs), networks, sensors, and weapons can be easily implemented and evaluated.

It should also be able to represent the behavior of entities, aggregates, and components. This should support a variety of mechanisms, including operator control of individual entities and aggregates, and semi-autonomous behavior, so that entities are capable of responding to events in a reasonable manner, even while engaged in carrying out other tasks. The general behavior model will be one of the most critical elements, as it will play a central role in establishing the cause-and-effect relationships that allow simulations to produce meaningful results.

The simulation framework should be capable of supporting multiple levels of detail, both in the representation of entities, and in the representation of their behavior. For example, aircraft movement should be able to be represented using linear interpolation between consecutive waypoints, spline-based interpolation between waypoints to produce continuous trajectories, and/or flight dynamics models of varying levels of complexity. Similarly, sensors, communications, and weapons models of varying levels of detail should be supported, and should be able to be combined with one another freely. This will require a carefully designed set of common interface standards.

Tools for developing scenarios will need to be able to access and exploit data sources such as the AODB within TBMCS for information on friendly force assets and plans, the MIDB for information on adversary forces, and environmental databases for information on terrain, weather, and other aspects of the environment. The MIDB fictional Pacifica databases provide a rich, complex environment to support the basic testing and demonstrating of a wide variety of prototype C4ISR tools and systems, while the real-world data in the MIDB can be used to support more advanced testing and demonstrations. The adversary modeling capabilities being developed by the Commander's Predictive Environment (CPE) program, and the environmental modeling capabilities being developed by the National Operational Environment Model (NOEM) program, should also be leveraged. The capabilities provided by JView for accessing environmental data should also be leveraged.

A change control board (CCB) consisting of representatives from multiple AFRL/RI divisions, branches, and sections, as appropriate, should be established to maintain consistency. Collaboration with the Air Force Agency for Modeling and Simulation (AFAMS), which is developing requirements for a new generation of Air Force simulations, is particularly critical.

The recommendations resulting from this effort are summarized below:

1) JSAF remains poorly suited for use within a small, dynamic laboratory environment. It requires a large, highly trained, and highly skilled staff to maintain, to operate, and especially to modify or extend it. Extending or modifying the JSAF software in support of a specific simulation experiment is difficult and time-consuming. Although it can be integrated with other existing tools and C4ISR systems, this is not easy to accomplish. Continued use of JSAF within AFRL/RI is therefore not recommended.

2) Detailed evaluations of more modern entity level simulations that support air operations should continue in a search for a better solution, as well as to identify requirements. Candidates include the Army's OneSAF Objective Objective System (OOS), as well as COTS simulations frameworks such as MaK Technologies VR-Forces and Ternion Corporation's FLAMES. However, these evaluations should be performed as separate, discrete tasks, each with a specific budget and time frame. A key part of the evaluation criteria should include how easy the simulation framework is to evaluate.

3) A meta-project should be created within AFRL/RI to develop an open, extensible simulation framework that can be used to support the widest possible variety of R&D projects in command and control, communications, sensor exploitation and management, and simulation science, with representatives from multiple divisions and branches, including those involved in all of these subject areas, across the air, space, and cyber domains. This project should leverage existing AFRL/RI models, databases, and other information sources to the maximum possible extent. It should be supported by a broad, diverse team of contractors, including representatives from academia, small businesses, and large businesses. It should initially focus on developing an architecture and a set of basic requirements for the simulation framework.

7 REFERENCED DOCUMENTS

1) NGA TM 8358.1, Datums, Ellipsoids, Grids, and Grid Reference Systems, 1990.

2) NGA TM 8358.2, The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS), 1989.

3) MIL-STD-2401, Department of Defense World Geodetic System (WGS), 1994.

4) STANAG 2211, Geodetic Datums, ellipsoids, grids and grid references, Fifth Edition, 1991.

5) TEC-SR-7, Handbook for Transformation of Datums, Projections, Grids and Common Coordinate Systems, 1996.

6) NGA, Department of Defense Glossary of Mapping, Charting, and Geodetic Terms, Fourth Edition, 1981.

7) Rapp, Richard H., Geometric Geodesy - Part I; Department of Geodetic Science and Surveying, The Ohio State University, Columbus, Ohio. 1984.

8) Rapp, Richard H., Geometric Geodesy - Part II; Department of Geodetic Science and Surveying, The Ohio State University, Columbus, Ohio. 1987.

9) Snyder, J. P., Geological Survey Professional Paper 1395 Map Projections - A Working Manual, 1987.

10) IEEE 1278.1A-1998 - Standard for Distributed Interactive Simulation - Application protocols.

11) Simulation Interoperability Standards Organization (SISO), Guide for: DIS Plain and Simple, SISO-REF-020-2007, 1 March 2007.

12) IEEE 1516-2000 - Standard for Modeling and Simulation High Level Architecture - Framework and Rules.

13) IEEE 1516.1-2000 - Standard for Modeling and Simulation High Level Architecture - Federate Interface Specification.

14) IEEE 1516.2-2000 - Standard for Modeling and Simulation High Level Architecture - Object Model Template (OMT) Specification.

15) IEEE 1516.3-2003 - Recommended Practice for High Level Architecture Federation Development and Execution Process (FEDEP).

16) SISO-STD-001.1-1999: Real-time Platform Reference Federation Object Model (RPR FOM 1.0).

17) DMO Focus Study, Cashulette C., Northrop Grumman, USAF AFRL Contract FA8750-06-C-0017, Aug 2008.

18) Automated Signal Identification and Modeling (ASIM), System Architecture and Design, Dr. Christopher Braun, ManTech International Corporation, August 2008.

8 ACRONYMS AND ABBREVIATIONS

AAA	Anti Aircraft Artillery
ABP	Air Battle Plan
AFRL	Air Force Research Laboratory
AOC	Air Operations Center
AODB	Air Operations Data Base
API	Application Programmer's Interface
ARP	Address Resolution Protocol
ASOC	Air Support Operations Center
AWACS	Airborne Warning and Control System
BACN	Battlefield Airborne Communications Node
BDA	Bomb Damage Assessment
BE	Basic Encyclopedia
BSD	Berkeley Software Distribution
C/JMTK	Commercial Joint Mapping Tool Kit
C3	Command, Control, and Communications
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance
CADRG	Compressed ARC Digital Raster Graphics
CALCM	Conventional Air Launched Cruise Missile
CAP	Combat Air Patrol
CCS	Coordinate Conversion Service
CD	Compact Disc
CDDL	Common Developer and Distribution Licence
CDLA	CWIN Data Logging and Analysis
CDRL	Contract Data Requirements List
CGF	Computer Generated Forces
CLIN	Contract Line
COA	Course of Action
COMINT	Communications Intelligence

CPL	Comm Parametric List (NSim)
COP	Common Operating Picture
CRC	Combat Reporting Center
CSAT	Coordinate System Analysis Team
CSV	Comma Separated Value
CTDB	Compact Terrain Data Base
CWIN	Cyber Warfare Integration Network
DARPA	Defense Advanced Research Projects Agency
DBST	Digital Battlestaff Sustainment Training
DCA	Defensive Counter Air
DCW	Digital Chart of the World
DIS	Distributed Interactive Simulation
DMO	Distributed Mission Operations
DMOTH	Distributed Mission Operations Test Harness
DMPI	Desired Mean Point of Impact
DMT	Distributed Mission Training
DoD	Department of Defense
DSAP	Dynamic Situation Awareness and Prediction
DTED	Digital Terrain Elevation Data
DTSim	Dynamic Terrain Simulation
DVD	Digital Versatile Disc
EBV	Entity Bit Vector
EC	Empire Challenge
EGM	Earth Gravity Model
ELINT	Electronics Intelligence
EPL	Emitter Parametric List (NSim)
ENU	East North Up
EOB	Enemy Order of Battle
FAARS	Future After Action Review System
FAT	Final Acceptance Test

FCR	Fire Control Radar
FEDEP	Federation Development and Execution Process
FLOMMR	Force Level Operational Mission Models, Rev 5
FOM	Federation Object Model
FTP	File Transfer Protocol
GB	Gigabyte
GBU	Guided Bomb Unit
GEOTRANS	Geographic Translator
GIE	Global Information Enterprise
GIESim	Global Information Enterprise Simulation
GMT	Greenwich Mean Time
GNU	GNU's Not Unix
GUI	Graphical User Interface
HLA	High Level Architecture
IADS	Integrated Air Defense System
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Standards Organization
ISR	Intelligence, Surveillance, and Reconnaissance
JBI	Joint Battlespace Infosphere
JFCOM	Joint Forces Command
JMTK	Joint Mapping Tool Kit
JNI	Java Native Interface
JRE	Java Runtime Environment
JSAF	Joint Semi-Automated Forces
JSB-RD	Joint Synthetic Battlespace for Research and Development
JSIMS	Joint Simulation System
JTAC	Joint Tactical Air Controller
JTT	Joint Targeting Toolbox

JUO	Joint Urban Operations
LGPL	Lesser General Public License
LOS	Line of Sight
M&S	Modeling and Simulation
MARCI	Multi-system Automated Remote Control and Instrumentation
MC02	Millenium Challenge 2002
METOC	Meteorological/Oceanographic
MGRS	Military Grid Reference System
MIDB	Military Intelligence Data Base
ModSAF	Modular Semi Automated Forces
MOVES	Modeling, Virtual Environments, and Simulation
MS	Microsoft
MSL	Mean Sea Level
MSP	Mensuration Services Program
MTDS	Modular Tactical Datalink Simulator
NCSFIE	Network-Centric SIGINT-Focused Information Enterprise
NGA	National Geospatial-Intelligence Agency
NSC	National Simulation Center
NSim	Network-centric Simulation
NSM	NCSFIE Simulation Messages
OASES	Ocean, Atmosphere, and Space Environmental Services
OMT	Object Model Template
OVCS	Orientation and Vector Conversion Service
PDU	Protocol Data Unit (DIS)
PFM	Pressure Field Modification
PPS	Points Per Second
PSM	Portable Space Model
PVD	Plan View Display
RARP	Reverse Address Resolution Protocol
RDBMS	Relational Data Base Management System

RH	Human Effectiveness Directorate (of AFRL)
RI	Information Directorate (of AFRL)
RID	RTI Initialization Data (HLA)
RISB	Decision Support Branch (of AFRL Information Directorate)
RMI	Remove Method Invocation (Java)
RPR	Real-time Platform Reference (FOM)
RTI	Run Time Infrastructure (HLA)
SAM	Surface to Air Missile
SE	Standard Edition (Java)
SEDRIS	Synthetic Environment Data Representation and Interchange Specification
SIGINT	Signals Intelligence
SIMPLE	Simulation to C4I Interchange Module for Plans Logistics and Exercises
SISO	Simulation Interoperability Standards Organization
SNE	Synthetic Natural Environment
SNN	Simulation Network News
SOF	Special Operations Forces
SOM	Simulation Object Model
SRM	Spatial Reference Model
SSR	Simulation Sensor (NSim)
STOW	Synthetic Theater of War
TADIL	Tactical Data Link
TADIL-J	Tactical Data Link - Joint
TAOS	Total Atmosphere Ocean Services
TBMCS	Theater Battle Management Core System
TCP	Transmission Control Protocol
TDAL	Targeting Database Access Layer
TEC	Topographic Engineering Center (US Army)
TEL	Transporter, Erector, Launcher
TENA	Test and Training Enabling Architecture
TM	Technical Manual

TMDB	Track Management Database
TNL	Target Nomination List
TR	Technical Report
TSO	Time Stamp Ordered
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UGS	Unattended Ground Sensors
UPS	Universal Polar Stereographic
USAF	United States Air Force
USSPACECOM	United States Space Command
UTC	Coordinated Universal Time
UTM	Universal Transverse Mercator
WGS	World Geodetic System
WOC	Wing Operations Center
XML	Extensible Markup Language

9 GLOSSARY

Distributed Modeling and Simulation Terms

Attribute - A named portion of an object state.

Attribute Ownership - The property of a federate that gives it the responsibility to publish values for a particular object attribute.

Cancellation - A mechanism used in optimistic synchronization mechanisms such as Time Warp to delete a previously scheduled event. Cancellation is a mechanism used within the Time Warp executive, and is normally not visible to the federate. It is implemented (in part) using the RTI's event retraction mechanism.

Causal Order - A partial ordering of messages based on the "causally happens before" ($\rightarrow$) relationship. A message delivery service is said to be causally ordered if for any two messages M_1 and M_2 (containing notifications of events E_1 and E_2, respectively) that are delivered to a single federate where $E_1 \rightarrow E_2$, then M_1 is delivered to the federate before M_2.

Class - A description of a group of objects with similar properties, common behavior, common relationships, and common semantics.

Class Hierarchy - A specification of a class-subclass, or "is-a" relationship between object classes in a given domain.

Component Class - An object class which is a component, or part of, a "composite" object which represents a unified assembly of many different object classes. The identification of a Component Class in the OMT should include cardinality information.

Conservative Synchronization - A mechanism that prevents a federate from processing messages out of time stamp order. This is in contrast to *optimistic* synchronization. The Chandy/Misra/Bryant null message protocol is an example of a conservative synchronization mechanism.

Constrained Simulation - A simulation where time advances are paced to have a specific relationship to wallclock time. These are commonly referred to as real-time or scaled-real-time simulations. Here, the terms *constrained simulation* and *(scaled) real-time simulation* are used synonymously. Human-in-the-loop (e.g., training exercises) and hardware-in-the-loop (e.g., test and evaluation simulations) are examples of constrained simulations.

Coordinated Time Advancement - A time advancement mechanism where logical clock advances within each federate only occur after some coordination is performed among the federates participating in the execution, e.g., to ensure that the federate never receives an event notice in its past. ALSP, for example, uses coordinated time advancement.

Current Time (of a federate) - Same as federate time.

Event - A change of object attribute value, an interaction between objects, an instantiation of a new object, or a deletion of an existing object that is associated with a particular point on the federation time axis. Each event contains a time stamp indicating when it is said to occur (also see definition of message).

Event Notice - A message containing event information.

Federate - A member of a HLA Federation. All applications participating in a Federation are called Federates. In reality, this may include Federate Managers, data collectors, live entity surrogates simulations, or passive viewers.

Federate Time - Scaled wallclock time or logical time of a federate, whichever is smaller. Federate time is synonymous with the "current time" of the federate. At any instant of an execution different federates will, in general, have different federate times.

Federation - A named set of interacting federates, a common federation object model, and supporting RTI, that are used as a whole to achieve some specific objective.

Federation Execution - The federation execution represents the actual operation, over time, of a subset of the federates and the RTI initialization data taken from a particular federation. It is the step where the executable code is run to conduct the exercise and produce the data for the measures of effectiveness for the federation execution.

Federation Object Model (FOM) - An identification of the essential classes of objects, object attributes, and object interactions that are supported by an HLA federation. In addition, optional classes of additional information may also be specified to achieve a more complete description of the federation structure and/or behavior.

Interaction - An explicit action taken by an object, that can optionally (within the bounds of the FOM) be directed toward other objects, including geographical areas, etc.

Interaction Parameters - The information associated with an interaction which objects potentially affected by the interaction must receive in order to calculate the effects of that interaction on it's current state.

Known Object - An object is known to a federate if the federate is reflecting or updating any attributes of that object.

Logical Time - A federate’s current point on the logical time axis. If the federate’s logical time is T, all time stamp ordered (TSO) messages with time stamp less than T have been delivered to the federate, and no TSO messages with time stamp greater than T have been delivered; some, though not necessarily all, TSO messages with time stamp equal to T may also have been delivered. Logical time does not, in general, bear a direct relationship to wallclock time, and advances in logical time are controlled entirely by the federates and the RTI. Specifically, the federate requests advances in logical time via the Time Advance Request and Next Event Request RTI services, and the RTI notifies the federate when it has advanced logical time explicitly through the Time Advance Grant service, or implicitly by the time stamp of TSO messages that are delivered to the federate. Logical time (along with scaled wallclock time) is used to determine the current time of the federate (see definition of federate time). Logical time is only relevant to federates using time stamp ordered message delivery and coordinated time advances, and may be ignored (by requesting a time advance to “infinity” at the beginning of the execution) by other federates.

Lookahead - A value used to determine the smallest time stamped message using the time stamp ordered service that a federate may generate in the future. If a federate's current time (i.e., federate time) is T, and its lookahead is L, any message generated by the federate must have a time stamp of at least T+L. In general, lookahead may be associated with an entire federate (as in the example just described), or at a finer level of detail, e.g., from one federate to another, or for a specific attribute. Any federate using the time stamp ordered message delivery service must specify a lookahead value.

Message - A data unit transmitted between federates containing at most one event. Here, a message typically contains information concerning an event, and is used to notify another federate that the event has occurred. When containing such event information, the message's time stamp is defined as the time stamp of the event to which it corresponds. Here, a "message" corresponds to a single event, however the physical transport media may include several such messages in a single "physical message" that is transmitted through the network.

Message (Event) Delivery - Invocation of the corresponding service (Reflect Attribute Values, Receive Interaction, Instantiate Discovered Object, or Remove Object) by the RTI to notify a federate of the occurrence of an event.

Model - A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process. [DoD 5000.59]

Object - A fundamental element of a conceptual representation for a federate that reflects the "real world" at levels of abstraction and resolution appropriate for federate interoperability. For any given value of time, the state of an object is defined as the enumeration of all its attribute values.

Object Model - A specification of the objects intrinsic to a given system, including a description of the object characteristics (attributes) and a description of the static and dynamic relationships that exist between objects.

Object Model Framework - The rules and terminology used to describe HLA object models.

Object Ownership - Ownership of the ID attribute of an object, initially established by use of the Instantiate Object interface service. Encompasses the privilege of deleting the object using the Delete Object service. Can be transferred to another federate using the attribute ownership management services.

Optimistic Synchronization - A mechanism that uses a recovery mechanism to erase the effects of out-of-order event processing. This is in contrast to *conservative* synchronization. The Time Warp protocol is an example of an optimistic synchronization mechanism. Messages sent by an optimistic federate that could later be canceled are referred to as optimistic messages.

Owned Attribute - An object attribute that is explicitly modeled by the owning federate. A federate that owns an attribute has the unique responsibility to provide values for that attribute to the federation, through the RTI, as they are produced.

Reflected Attribute - An object attribute that is represented but not explicitly modeled in a federate. The reflecting federate accepts new values of the reflected attribute as they are produced by some other federation member and provided to it by the RTI.

Retraction - An action performed by a federate to unschedule a previously scheduled event. Event retraction is visible to the federate. Unlike "cancellation" that is only relevant to optimistic federates such as Time Warp, "retraction" is a facility provided to the federate. Retraction is widely used in classical event oriented discrete event simulations to model behaviors such as preemption and interrupts.

RTI Initialization Data (RID) - The data required by the RTI for operation. The required data come from two distinct sources, the Federation Object Model (FOM) product, and the Federation Required Execution Details (FRED).

Runtime Infrastructure (RTI) - The general purpose distributed operating system software which provides the common interface services during the runtime of an HLA federation.

Simulation - A method for implementing a model over time. Also, a technique for testing, analysis, or training in which real-world systems are used, or where real-world and conceptual systems are reproduced by a model. [DoD 5000.59]

Simulation Object Model (SOM) - A specification of the intrinsic capabilities that an individual simulation offers to federations. The standard format in which SOMs are expressed provides a means for federation developers to quickly determine the suitability of simulation systems to assume specific roles within a federation.

Time Flow Mechanism - The approach used locally by an individual federate to perform time advancement. Commonly used time flow mechanisms include event driven (or event stepped), time driven, and independent time advance (real-time synchronization) mechanisms.

Time Management - A collection of mechanisms and services to control the advancement of time within each federate during an execution in a way that is consistent with federation requirements for message ordering and delivery.

Time Stamp (of an Event) - A value representing a point on the federation time axis that is assigned to an event to indicate when that event is said to occur. Certain message ordering services are based on this time stamp value. In constrained simulations, the time stamp may be viewed as a deadline indicating the latest time at which the message notifying the federate of the event may be processed.

Time Stamp Order (TSO) - A total ordering of messages based on the "temporally happens before" ($\rightarrow_t$) relationship. A message delivery service is said to be time stamp ordered if for any two messages M_1 and M_2 (containing notifications of events E_1 and E_2, respectively) that are delivered to a single federate where $E_1 \rightarrow_t E_2$, then M_1 is delivered to the federate before M_2. The RTI ensures that any two TSO messages will be delivered to all federates receiving both messages in the same relative order. To ensure this, the RTI uses a consistent tie-breaking mechanism to ensure that all federates perceive the same ordering of events containing the same time stamp. Further, the tie-breaking mechanism is deterministic, meaning repeated executions of the federation will yield the same relative ordering of these events if the same initial conditions and inputs are used, and all messages are transmitted using time stamp ordering.

Transportation Service - An RTI provided service for transmitting messages between federates. Different categories of service are defined with different characteristics regarding reliability of delivery and message ordering.

True Global Time - A federation-standard representation of time synchronized to GMT or UTC (as defined in this glossary) with or without some offset (positive or negative) applied.

Unconstrained Simulation - A simulation where there is no explicit relationship between wallclock time and the rate of time advancements. These are sometimes called “as-fast-as-possible” simulations, and these two terms are used synonymously here. Analytic simulation models and many constructive “war game” simulations are often unconstrained simulations.

Wallclock Time - A federate's measurement of true global time, where the measurand is typically output from a hardware clock. The error in this measurement can be expressed as an algebraic residual between wallclock time and true global time or as an amount of estimation uncertainty associated with the wallclock time measurement software and the hardware clock errors.

Coordinate Conversison and Datum Transformation Terms

Coordinate – Linear or angular quantities that designate the position that a point occupies in a given reference frame or system. Also used as a general term to designate the particular kind of reference frame or system, such as Cartesian coordinates or spherical coordinates.

Datum – Any numerical or geometrical quantity or set of such quantities specifying the reference coordinate system used for geodetic control in the calculation of coordinates of points on the earth. Datums may be either global or local in extent. A local datum defines a coordinate system that is used only over a region of limited extent. A global datum specifies the center of the reference ellipsoid to be located at the earth's center of mass and defines a coordinate system used for the entire earth.

Elevation – Vertical distance measured along the local plumb line from a vertical datum, usually mean sea level or the geoid, to a point on the earth.

Ellipsoid – The surface generated by an ellipse rotating about one of its axes. Also called ellipsoid of revolution.

Equator – The line of zero geodetic latitude; the great circle described by the semi-major axis of the reference ellipsoid as it is rotated about the semi-minor axis.

Equipotential Surface – A surface with the same potential, usually gravitational potential, at every point; a level surface.

Geocentric Coordinates – Cartesian coordinates (X, Y, Z) that define the position of a point with respect to the center of mass of the earth.

Geodetic Coordinates (Geodetic Position) – The quantities of latitude, longitude, and geodetic height (ϕ, λ, h) that define the position of a point on the surface of the earth with respect to the reference ellipsoid.

Geodetic Height (Ellipsoid Height, h) – The height above the reference ellipsoid, measured along the ellipsoidal normal through the point in question. The geodetic height is positive if the point is outside the ellipsoid.

Geodetic Latitude (ϕ) – The angle between the plane of the Equator and the normal to the ellipsoid through the computation point. Geodetic latitude is positive north of the equator and negative south of the Equator.

Geodetic Longitude (λ) – The angle between the plane of a meridian and the plane of the prime meridian. A longitude can be measured from the angle formed between the local and prime meridians at the pole of rotation of the reference ellipsoid, or by the arc along the Equator intercepted by these meridians.

Geoid – The equipotential surface in the gravity field of the Earth that approximates the undisturbed mean sea level extended continuously through the continents. The geoid is the surface of reference for astronomic observations and geodetic leveling. Orthometric heights are referred to the surface of the geoid.

Geoid Separation (N) – The distance between the geoid and the mathematical reference ellipsoid as measured along the ellipsoidal normal. This distance is positive outside, or negative inside, the reference ellipsoid. Also called geoidal height; undulation of the geoid.

Grid Reference System – A plane-rectangular coordinate system usually based on, and mathematically adjusted to, a map projection in order that geodetic positions (latitudes and longitudes) may be readily transformed into plane coordinates and the computations relating to them may be made by the ordinary methods of plane surveying.

Horizontal Datum – A horizontal datum specifies the coordinate system in which latitude and longitude of points are located. The latitude and longitude of an initial point, the azimuth of a line from that point, and the semi-major axis and flattening of the ellipsoid that approximates the surface of the earth in the region of interest define a horizontal datum.

Map Projection – A function relating coordinates of points on a curved surface (usually an ellipsoid or sphere) to coordinates of points on a plane. A map projection may be established by analytical computation or, less commonly, may be constructed geometrically.

Map Scale – The ratio between a distance on a map and the corresponding actual distance on the earth's surface.

Mean Sea Level (MSL) – The average height for the surface of the sea for all stages of the tide, used as a reference for elevations. Also called Sea Level Datum. Mean Sea Level and the surface of the geoid are often assumed to coincide though in reality they are approximations to one another and can be offset by meters in some locations. GEOTRANS computes orthometric heights. However, because the MSL surface and the geoid surface are good approximations to each other in many locations, the more commonly used term MSL height is used to refer to these heights.

Meridian – A north-south reference line, particularly a great circle through the geographical poles of the earth, from which longitudes and azimuths are determined; or the intersection of a plane forming a great circle that contains both geographic poles of the earth, and the ellipsoid.

MSL Height – An elevation or height referenced to mean sea level.

Orthometric Height – The distance of a point from the geoid measured along the direction of gravity at that point, with heights of points outside the geoid being treated as positive.

Parallel – A line on the earth, or a representation thereof, which represents the same latitude at every point.

Prime Meridian – A meridian from which the longitudes of all other meridians are reckoned. This meridian, of longitude 0°, was traditionally chosen to pass through the Greenwich Observatory in Greenwich, England. For new refined coordinate systems, the location of the prime meridian is defined by the International earth Rotation Service, Paris, France.

Reference Ellipsoid – An ellipsoid, usually a bi-axial ellipsoid of revolution, whose dimensions closely approach the dimensions of the geoid; the exact dimensions are determined by various considerations of the section of the earth's surface concerned. .

Scale Factor (Projection) – A multiplier for reducing a distance in a map projection to the actual distance on the chosen reference ellipsoid.

Vertical Datum – A vertical datum is the surface to which elevations are referenced. A local vertical datum is a continuous surface, usually mean sea level, at which elevations are assumed to be zero throughout the area of interest.